M62/cM62 Hardware Manual

The M62/cM62 Hardware Manual was prepared by the technical staff of Innovative Integration, December 1997
For further assistance contact

Innovative Integration
5785 Lindero Canyon Road
Westlake Village, California 91362

PH:(818) 865-6150
FAX:(818) 879-1770
email:techsprt@innovative-dsp.com
WWW:www.innovative-dsp.com

VSS\M62\documents\hardware\M62hdw.fm

#51049 rev – 1.05

CHAPTER 1

CHAPTER 2

List of Tables

List of Figures

CHAPTER 1

M62/cM62 Hardware Manual

M62/cM62 Hardware Functions

The M62 is a PCI bus compatible digital signal processor (DSP) card based around the Texas Instruments TMS320C6201 processor. Implementing a modular I/O expansion system, the M62 is particularly suited to data acquisition and control tasks, and is supported by a collection of I/O bus function cards which provide hardware interfacing to real-world equipment.

The M62's features include:

1. TMS320C6201 processor
2. Optional external zero wait-state SBSRAM and one wait-state SDRAM memory pools
3. Two inter-board communications ports (up to 80 Mbytes/sec transfer rate)
4. Six channels of on-board timing (two on-chip timers, three custom 16-bit timers in FPGA logic and the 9850 DDS timebase)
5. OMNIBUS module compatible (two available slots on M62, three on cM62)
6. 32 bits of digital I/O
7. Two serial port connectors
8. External mux board control connectors (compatible with external TERM multiplexer and signal conditioner boards)

9. JTAG hardware emulation support

The following figure gives a block diagram of the M62/cM62.

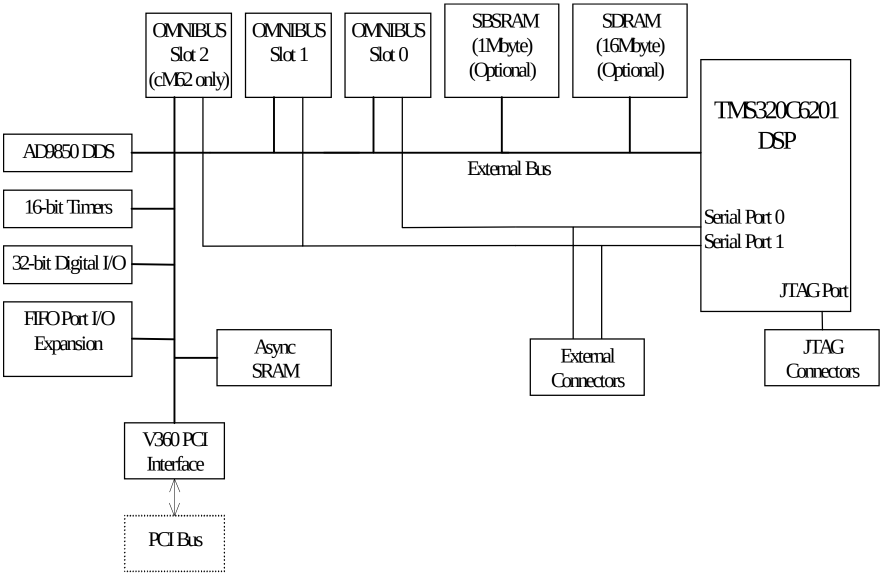


FIGURE 1. **M62/cM62 Block Diagram**

The cM62 is a Compact PCI compatible version of the M62 board. The cM62 retains all of the features of the M62 but is intended for use in Compact PCI host systems. In addition to the M62 feature set, the cM62 includes an additional OMNIBUS slot (allowing up to three OMNIBUS modules to be installed on a single cM62 board).

For brevity's sake this manual will refer to both cards as the M62. Any differences in functionality between the two boards (support of the third I/O bus site, different types of connectors, etc.) will be noted as required. In addition, the PCI and Compact PCI buses are collectively referred to as the PCI bus.

Memory Map

The M62 processor operates in ‘C6201 HPI boot mode with memory map type 1. In this mode, the processor’s memory is available to the PCI host computer via the processor’s host port interface (HPI). On-chip memory is mapped starting at address 0. Applications programs are loaded via the HPI by the host while the card is in reset: once the program is loaded, reset is deasserted by the host and the processor boots from on-chip RAM starting at address 0.

The following figure gives the processor memory map of the M62 for external peripherals and memory. Please note that this table ignores any on-chip resources.

Function	**Address**	**Memory Space**
FIFOPort	0x400000	CE0
V360 Registers	0x1400000	CE1
FIFOPort Reset	0x1410000	
FIFOPort Enable	0x1420000	
OMNIBUS Control (reserved)	0x1430000	
External Mux Control 0	0x1440000	
External Mux Control 1	0x1450000	
AD9850 Reset	0x1470000	
AD9850 Frequency Update	0x1480000	
AD9850 Write Clock	0x1490000	
Digital I/O Data Register	0x14A0000	
Digital I/O Direction Control	0x14B0000	
Digital I/O Input Latch Clock Control Register	0x14C0000	
Transmit FIFOPort PEN* Mode	0x14D0000	
Receive FIFOPort Level Status	0x14D4000	
Transmit FIFOPort Level Status	0x14D8000	
16 bit PIT Timers	0x14F0000	
External Interrupt Input 4 Select	0x1500000	
External Interrupt Input 5 Select	0x1510000	
External Interrupt Input 6 Select	0x1520000	
External Interrupt Input 7 Select	0x1530000	
OMNIBUS Strobe 0	0x1540000	
OMNIBUS Strobe 1	0x1550000	
OMNIBUS Strobe 2	0x1560000	
OMNIBUS Strobe 3	0x1570000	
OMNIBUS Strobe 4	0x1580000	
OMNIBUS Strobe 5	0x1590000	
OMNIBUS Strobe 6	0x15A0000	
OMNIBUS Strobe 7	0x15B0000	
OMNIBUS Strobe 8 (cM62 only)	0x15C0000	
OMNIBUS Strobe 9 (cM62 only)	0x15D0000	
OMNIBUS Strobe 10 (cM62 only)	0x15E0000	
OMNIBUS Strobe 11 (cM62 only)	0x15F0000	
Async SRAM (128Kx32)	0x1600000	

SDRAM (16Mbyte) (optional)	0x2000000	CE2
SBSRAM (1Mbyte) (optional)	0x3000000	CE3

TABLE 1. M62 External Memory Map

M62 Hardware Initialization Requirements

The M62 design requires the following values to be written to its hardware control registers in order to provide access to on-board hardware:

Register	**Address**	**Value**
EMIF Global Control	0x01800000	0x00003069
CE1 Control	0x01800004	0x73E70F22
CE0 Control	0x01800008	0x11010410
CE2 Control	0x01800010	0x00000030
CE3 Control	0x01800014	0x00000040
SDRAM Control	0x01800018	0x07117000
SDRAM Refresh	0x0180001C	0x00000618
Interrupt Polarity	0x019C0008	0x0000000F

TABLE 2. M62 Bus Control Register Initialization Values

These values are initialized automatically by C programs compiled under the M62 Development Package software libraries. Be sure to include initialization of these values whenever software is developed outside the Development Package or when a JTAG hardware assisted debugger is employed for code downloading to the M62 (i.e. when using Code Composer or any other JTAG debugger package).

External Memory

The M62 offers three types of external memory: asynchronous SRAM (ASRAM), synchronous DRAM (SDRAM), and synchronous burst SRAM (SBSRAM). The 128Kx32 ASRAM memory comes standard with the M62, while the SBSRAM and SDRAM are optional.

ASRAM is used by the M62 as a buffer for bus master and slave data movement on the PCI bus. The ASRAM is accessible by the V360 PCI bus interface device, allowing the processor to setup bus master data transfers which are handled as a

DMA-style transfer by the V360. The M62 utilizes the ‘C6201’s HOLD/HOLDA bus grant feature to provide ASRAM access to the V360. In addition, the ASRAM memory acts as a target for slave accesses by other PCI bus masters (either the host processor or other adapter cards).

The optional SBSRAM and SDRAM memories provide large, fast areas to store copious amounts of data or program information. The SBSRAM and SDRAM memories are not accessible by the PCI interface.

The ‘C6201 processor operates in big endian addressing mode, allowing 8, 16, and 32 bit wide data movement to and from external SBSRAM and SDRAM memory. Async SRAM supports 32-bit accesses only, as does the V360 PCI bus interface.

M62 OMNIBUS

The M62 I/O bus provides a modular, high-speed expansion area which is directly tied to the processor’s bus and which is ideally suited for I/O hardware expansion. Direct memory-mapped accesses allow the processor to transfer data to and from I/O bus peripherals constructed as plug-in modules which can be mixed and matched to suit the particular user’s functional requirements.

The OMNIBUS slots are accessed as memory-mapped peripherals with the M62 providing four decoded chip select signals per slot. The following figure gives the memory map for the OMNIBUS slots, and shows the decode signal to slot mapping.

TABLE 3. M62 I/O Bus Memory Mapping

Function	**Starting Address**	**Module Slot**
OMNIBUS Strobe 0	0x1540000	0
OMNIBUS Strobe 1	0x1550000	0
OMNIBUS Strobe 2	0x1560000	0
OMNIBUS Strobe 3	0x1570000	0
OMNIBUS Strobe 4	0x1580000	1
OMNIBUS Strobe 5	0x1590000	1
OMNIBUS Strobe 6	0x15A0000	1
OMNIBUS Strobe 7	0x15B0000	1
OMNIBUS Strobe 8 (cM62 only)	0x15C0000	2
OMNIBUS Strobe 9 (cM62 only)	0x15D0000	2
OMNIBUS Strobe 10 (cM62 only)	0x15E0000	2
OMNIBUS Strobe 11 (cM62 only)	0x15F0000	2

Each module site provides a 32-bit wide data bus connection to the processor’s data bus, with 12 bits of low-order address signals for additional decoding beyond the four chip select signals available per slot. Each module also connects to a ‘C6201 serial port (serial port zero for slot zero, and serial port 1 for slots 1 and 2) to allow serial port driven I/O. Bus reset, RDY, R/W, and processor clock signals are available, as are power connections for digital 5V and analog +-5V and +-15V. Timebase connections include timer channels from both the custom 16-bit timers and the 9850 direct-digital synthesizer.

Each OMNIBUS slot has a 50 pin undedicated connector (JP17 on slot 0, JP21 on slot 1, and JP32 on slot 2) for use in providing external I/O to/from a module installed in the slot. The slot’s I/O connector is in turn pinned out to a 50 pin .100” square double row header (JP18 for slot 0, JP22 for slot 1) on the M62 and to 50 pin mini SCSI style connectors on the cM62 (JP18 for slot 0, JP22 for slot 1, and JP33 for slot 2). The M62 also provides 15 pin external connectors for each slot which allow the highest numbered 15 signals on the header conectors to be pinned out external to the host computers chassis.

Connector pinouts for the module sites are provided in the appendices. Individual pin functions are noted in the tables, and in general the OMNIBUS pinout represents a direct connection to the ‘C6201 local bus.

M62 OMNIBUS Memory Mapping

Since the ‘C6201 processor is a byte addressable machine which implements its address bus based on a 32-bit transfer width (i.e. the address bus starts at A2 and separate byte enable pins are supplied to control accesses to individual bytes within the 32-bit wide location denoted by the address bus), users must take care when writing software which performs OMNIBUS accesses.

The OMNIBUS specification requires 32-bit accesses and does not support byte or half-word (16-bit) accesses. No support is included in the specification for the ‘C6201’s byte enable pins. This means that software performing accesses must always perform 32-bit transactions with the OMNIBUS modules. When writing C code for the M62, programmers should use only variables of type int or unsigned int (or their derived types), and all accesses should be word justified (the least significant nibble of the address must always be a multiple of four). Accesses generated using pointers to variables of type char, short, or long will cause erroneous non-32-bit accesses. Correct OMNIBUS module operation under these situations is not guaranteed.

Please note also that memory decoding within the OMNIBUS decode regions uses 32-bit addressing and that the memory map tables given in the *OMNIBUS Hardware Manual* should be treated appropriately. For example, the description of the OMNIBUS DIG module notes that the byte 3 direction control register for a module installed in site 0 is mapped to address IOMOD2 + 3. This address should be literally interpreted as 0x156000C, where IOMOD2 is equal to 0x1560000 and the offset adds decimal 12 (three 32-bit words of offset). IOMOD2 + 3 should NOT be interpreted as 0x1560003, since the offset is 3 32-bit words, not 3 bytes.

This addressing is most easily handled in C by using integer pointers and integer pointer arithmetic, which will always result in the required address alignment. For example, the following code defines a pointer and accesses the byte 3 direction control register with the documented offset:

```
unsigned int *pointer = 0x1560000;

*(pointer + 3) = 0x0;        /* set byte 3 to output mode */
```

The actual accessed memory location is 0x156000C, due to the way pointer math is handled in C.

OMNIBUS Power

The OMNIBUS interface provides five separate power supplies for use by modules along with two separate ground return connections. The following table lists the supplies and their power ratings. A separate digital 5V supply is provided along with separate digital grounds to minimize the digital noise present on the analog power supplies.

Pin Name	Voltage	Current Rating (max)
DVCC	5V (digital)	(System dependent)
+12	12V	(System dependent)
-12	-12V	(System dependent)
AVCC	5V (analog)	500 mA
-AVCC	-5V	500 mA
+AV	+15V	330 mA
-AV	-15V	330 mA

TABLE 4. **I/O Bus Power Ratings**

Please note that the AGND and DGND busses are separated on the M62 and for proper ground referencing they must be tied together on modules which use the analog power supplies (any supply other than digital 5V, 12V, or –12V). Innovative Integration recommends that a ferrite bead (Panasonic EXC-ELSA35V or equivalent) be used on custom modules to connect the two ground busses in order to prevent high frequency digital noise on the DGND bus from polluting the clean AGND return.

FIFOPort I/O Expansion

The FIFOPort feature provides a buffered bidirectional 16 bit interface which allows external hardware or other M62 boards to communicate with the M62 at high data rates. A single input FIFO is provided to buffer incoming strobed parallel data, while a FIFOPort compatible output supports clocking data to external hardware or other FIFOPorts. Access to the ‘C6201 timer I/O pins is provided to support simple bit I/O requirements.

The following diagram illustrates the FIFOPort’s operation. The FIFO buffer memory serves to clock incoming data and store it for use by the ‘C6201. Data is formatted as a 16 bit wide data bus synchronous with an rising edge strobe signal which acts as the FIFO load clock. The output portion consists of the same two signals: output data plus the strobe signal for the receiving end of the port.

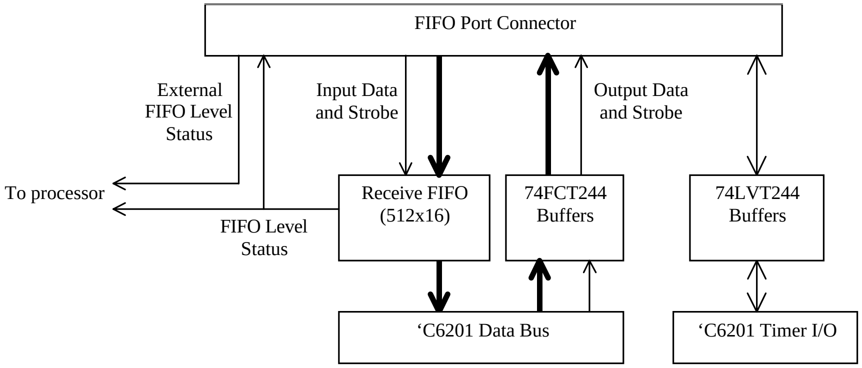


FIGURE 2. **FIFOPort Block Diagram**

The FIFOPort also provides external access to the receive FIFO’s empty, full, and programmable almost full flags to allow hardware to monitor the FIFO’s level status. The port can also receive FIFO level status from external hardware to allow the ‘C6201 processor to monitor level status of FIFOs located off the M62 card. Both the onboard receive FIFO level status and the off board FIFO status lines may be polled or may generate interrupts to the ‘C6201 processor.

In addition to the FIFO data management functions, access to the ‘C6201 timer I/O pins is provided to support simple bit I/O requirements. The timer I/O pins are buffered through LVT family logic buffers and driven on/off the card for use where individual bit I/O control is needed to external hardware.

Also available on the FIFOPort connector is an external interrupt input which is connected to the processor’s interrupt switch matrix and which allows the ‘C6201 to receive an active low interrupt from external hardware.

Transmitting and Receiving FIFOPort Data

Data is transmitted and received on the FIFOPort by means of processor address location 0x400000. EMIF read and write accesses (either due to CPU or DMA activity) cause read and write strobes to be generated to the FIFOPort circuitry only when this address is accessed.

In the case of a write access, an active high output strobe is generated on the external connector and 16 bit bus data is driven out to the output bits. This data should be latched by external hardware on the rising edge of the FIFOPort output strobe. Write accesses do not affect the current state of the receive FIFO.

In the case of a read, an input read strobe is generated to the receive FIFO and its output data latched by the processor. If the data item being read in the current cycle is not the last item stored in the buffer, the next data item is clocked out by the FIFO and held ready for the next read access by the processor. Read accesses do not generate output strobes to the external connector.

Please note that the data returned by the FIFO on a read access is present on the least significant 16 bits of the processor’s data bus. The most significant 16 bits are not driven and are not defined. If 32-bit CPU accesses are being used to read data from the FIFO, then the upper 16 bits of the result should be masked off before use. The DMA controller may be programmed for 16 bit access width and will automatically perform 16-bit to 32-bit data translation such that each stored 32-bit wide data item retrieved will be the concatenation of two 16-bit values read from the FIFO.

If the receive FIFO grows empty, the last data item's value will be output on any subsequent read accesses.

Monitoring FIFO Status

The FIFOPort provides a FIFO level monitoring feature which allows software to read the receive FIFO's level indicators as well as FIFO level data from external hardware (if connected). The receive FIFO's empty, full, and programmable almost full flags can be read at any time by the CPU, or the interrupt selection matrix may be programmed to notify the CPU of level events via an interrupt (see Interrupts section for more information). The same functionality is provided for the external FIFO, allowing the CPU to read back or be interrupted by any of six different level state conditions.

The FIFO level status is monitored using tweo registrs, one for the receive FIFO and one for the transmit FIFO (if connected). The register bit definitions are given below.

Bit Number:	31-4	3	2	1	0
Bit Field:	Reserved	RCV_AF	RCV_FULL	RCV_HF	RCV_EMPTY

FIGURE 3. **Receive FIFOPort Level Status Register**

Bit Field Name	**Function**
RCV_EMPTY	Receive FIFO Empty Flag (1 = empty, 0 = not empty)
RCV_HF	Receive FIFO Half-full Flag (1 = not half full, 0 = at least half full)
RCV_FULL	Receive FIFO Full Flag (1 = not full, 0 = full)
RCV_AF	Receive FIFO Almost-full Flag (1 = almost-full, 0= not almost-full)

TABLE 5. **Receive FIFOPort Level Status Register Definition**

Bit Number:	31-4	3	2	1	0
Bit Field:	Reserved	TX_AF	TX_FULL	TX_HF	TX_EMPTY

FIGURE 4. **Transmit FIFOPort Level Status Register**

Bit Field Name	**Function**
TX_EMPTY	Transmit FIFO Empty Flag (0 = empty, 1 = not empty)
TX_HF	Transmit FIFO Half-full Flag (0 = not half full, 1 = at least half full)
TX_FULL	Transmit FIFO Full Flag (1 = not full, 0 = full)
TX_AF	Transmit FIFO Almost-full Flag (1 = almost-full, 0= not almost-full)

TABLE 6. **Transmit FIFOPort Level Status Register Definition**

The receive FIFO level bits are read directly from the FIFO hardware on the corresponding FIFOPort, while the transmit FIFO bits are read from the level input pins on the FIFOPort connector. If no external status is being reported by the hardware connected to the FIFOPort, then these bits will read as ones (onboard 10K pullup resistors hold the transmit input pins high). If external FIFO level reporting is not desired, the level inputs may be used for application specific bit inputs to report other hardware status conditions or trigger interrupts on the M62 processor (note that this is in addition to the dedicated timer I/O pins and the processor interrupt input pin on the FIFOPort connector, which remain available regardless of the use of the FIFO status inputs). The digital return levels given for the transmit FIFO assume connection to another ‘C6x card manufactured by Innovative Integration, or to hardware emulating similar FIFO level reporting.

By appropriate programming, the FIFO levels may also be monitored using processor interrupts. The three status bits for each FIFO in each direction are available as sources to the interrupt selection matrix for each processor. This technique is typically used to drive DMA transfers to and from the FIFOPort, where one FIFO status interrupt triggers one or more transfers using DMA synchronization, or for CPU interrupts where the target CPU in a transfer wants to be interrupted when data (or space) is available in the FIFO (this would be typical of “one-shot” FIFO transfers, where a single full FIFO’s worth of data is transferred at once and the receiving processor needs to be notified when the FIFO reached the full state so that a read operation on the other side of the FIFO may commence). For more information on using the FIFO levels to trigger interrupts to the ‘C6201 processors, see the Interrupts section.

FIFOPort Reset

The receive FIFO may be cleared and its condition reset at any time by accessing the FIFOPort reset register at address 0x1410000. The data written to the register is not critical: a write access of any data will reset in the FIFO being reset. Upon reset, the FIFO levels are cleared and the flags change to reflect the FIFO empty status, and the programmable almost full control variables are reset to default values (see below for more information).

FIFOPort Enable

After a board reset or power up and prior to reading data from the receive side of the FIFOPort, software must enable data output by accessing the FIFOPort enable register at address 0x1420000. Either a read or write access to the register may be used to enable the FIFOPort. Data reads issued to the FIFOPort prior to enabling the port will clock buffered data out of the port (if any data is stored in the FIFO) but the data will not be read correctly by the processor.

Controlling the FIFOPort Programmable Almost-full Flag

In addition to the fixed function empty and full flags, the FIFOPort provides a programmable almost-full flag which can be used to enable notification on partial FIFO transfer lengths. This feature is particularly suitable to DMA block transfers on the FIFOPort because it maximizes the transfer rates on both sides of the FIFO by keeping the buffer partially filled.

The almost-full flag operates as follows: given two initialization bytes (X and Y), the FIFO outputs an almost-full/almost-empty flag function which is active whenever the FIFO contains X or less words of data or 512-Y or more words of data. By programming the X value equal to the almost-full level and the Y value to zero, the FIFO’s programmable flag effectively becomes a variable partial full indicator. For example, programming the X variable to 128 and the Y variable to 0 yields a quarter-full output function.

The programmable almost-full flag control variables for the transmit half of the FIFOPort are initialized by enabling PEN mode, then writing the variables to the

FIFOPort. PEN mode is enabled by writing a zero to the transmit FIFOPort PEN mode register at address 0x14D0000. The X variable is then written to the FIFOPort, followed by the Y variable, with both data values being 8 bits wide and right justified on the bus. The default values for the X and Y variables are both 64 (the FIFO reverts back to these values on a reset). Following the completion of the Y variable write, PEN mode should be disabled by writing a 1 to the PEN mode register. Please note that the almost-full flag variables may only be written immediately after a FIFO reset has been issued to the transmit side FIFO and before any data is written to the transmit FIFO.

Note: the above description of the PEN mode register operation was a change to the M62 control logic made in April 1999. Boards purchased earlier than this date should be returned to Innovative for an update. Please contact Innovative with questions concerning this feature.

Please note that this initialization operation only affects the transmit FIFO (i.e. the FIFO on the external hardware or other M62 or Quatro62 card). The FIFOPort architecture does not allow the onboard processors to initialize the programmable levels of the FIFOPort receive FIFOs. This initialization is always performed by the external hardware prior to writing data to the receive FIFO.

Timer I/O and the FIFOPort

The FIFOPort also provides a connection to the processor’s timer I/O pins. This allows designers of hardware connecting to the FIFOPort easy access to four bits of unidirectional I/O for control purposes and status reporting. The on-chip timers of the ‘C6201 may be programmed to generate or receive clock and count events on the pins, or the pins may be used for general purposes bit I/O.

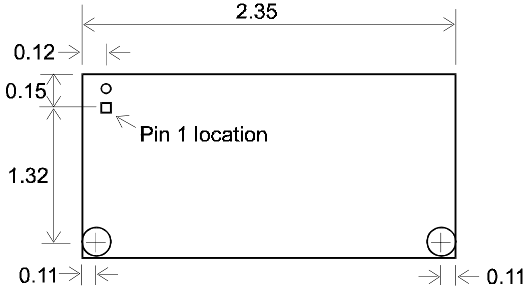
The M62 implements LVT family buffering between the timer I/O pins and the FIFOPort connector. Output and input levels are TTL compatible, but the outputs will not drive beyond 3.3V on the high side, and are tolerant of input voltages of up to 5V. This feature makes the FIFOPort timer I/O pins suitable for direct interfacing to 3.3V or 5V TTL compatible logic (such logic families as HCT, LSTTL, FCT, ABT, and ACT may be directly connected to the FIFOPort timer I/O pins).

Designing External Hardware for use with the FIFOPort

Use caution when designing external hardware which is to be connected to the FIFOPort. The signals present on the interface connector are extremely high speed and failure to handle them appropriately can cause functional problems with the FIFOPort as well as the M62's onboard components. Innovative does not recommend driving cables directly as capacitive load and ringing issues can cause corruption of the transmitted data. FIFOPort connector pinouts are given in the appendices.

The M62 provides mechanical mount holes suitable for use in attaching daughterboard style printed circuit boards to the FIFOPort connector. The combination of the FIFOPort connector retention and the mount hole positioning allow designers to easily create interface modules for use in adapting the FIFOPort connector to external hardware.

The following diagram gives mechanical dimensions for a FIFOPort compatible daughter PC board.



Mount hole = 0.115 inch (#4)
All Measurements in Inches
Not to Scale
+/-0.005 inch tolerance

FIGURE 5. **FIFOPort Daughterboard Mechanical Dimensions**

FIFOPort Timing

The following diagrams give timing information for the FIFOPort circuitry. This data is derived from device specifications and is not factory tested.

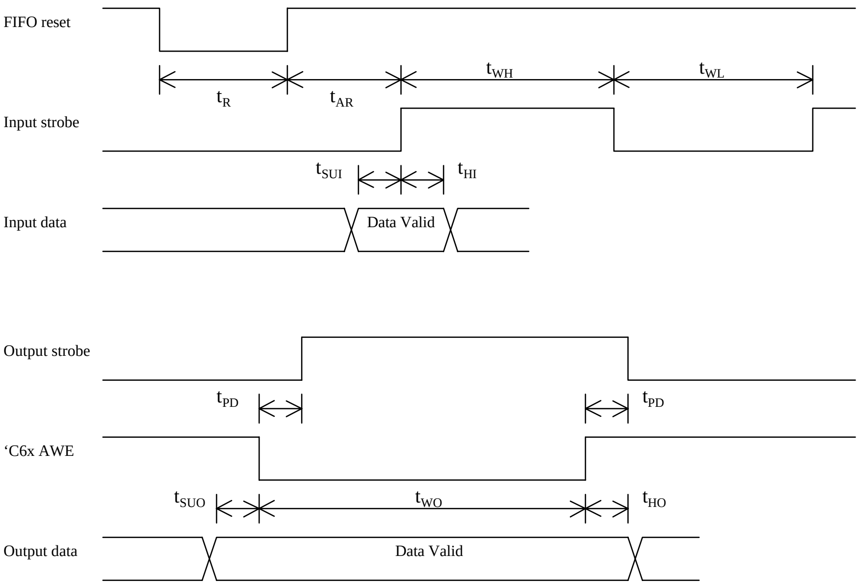


FIGURE 6. FIFOPort Timing

Parameter	min (ns)	max (ns)
t_{WH}	7	
t_{WL}	7	
t_{SUI}	5	
t_{HI}	0	
t_R	10	
t_{AR}	5	
t_{PD}		7
t_{SUO}	10[1]	
t_{HO}	0[1]	
t_{WO}	10[1]	

TABLE 7. FIFOPort Timing Parameters

Notes [1]: Dependent on EMIF programming for CE0 space as well as processor cycle frequency. These values are determined from recommended EMIF register values.

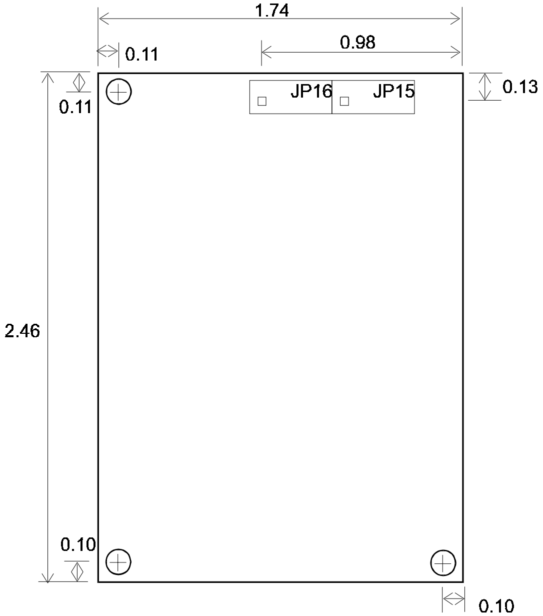
Serial Ports

The ‘C6201’s on-chip serial ports are pinned out to connectors JP15 (port 0) and JP16 (port 1) for use with external hardware. The serial ports are also connected to the OMNIBUS slots for use with modules designed to interface to the processor serially.

Pinouts for the serial port connectors are given in the appendicies. Innovative recommends buffering these ports with off board hardware in order to preserve signal integrity.

The following diagram shows the mechanical dimensions for a suggested printed circuit board outline for use in providing buffering of serial bus signals to external hardware.

FIGURE 7. **Serial Port Daughterboard Mechanical Dimensions**



Top view, serial port connectors facing down (mounted on opposite face of board)
Mount hole = 0.115 inch (#4)
Connectors arrayed at 2mm spacing (pin 1 locations shown)
All Measurements in Inches
Not to Scale
+/-0.005 inch tolerance

Timers

The M62 provides a total of six channels of independent timebase generation on board for use in timing data acquisition, servo controls, real-time counters, and other applications. The functionality is divided into three devices: two 32-bit timer channels on the ‘C6201 processor, three custom 16-bit timer channels in external logic, and a 32-bit direct digital synthesizer (DDS) channel in the AD9850 device. This section discusses the AD9850 and external timers: for more information on the on-chip timers, see the *TMS320C6201 Peripherals Reference Guide*.

On-chip Timers

The on-chip timers are available for use as software timebases and interrupt generators, and are pinned out to connector JP31 for use with external hardware. All four processor timer pins are available on JP31, allowing applications to use each timer as a time base output, an event counter input, or as bit I/O. The timer pins are also available on the FIFOPort connector for use in controlling external hardware attached to the FIFOPort. See the appendix for pinout descriptions.

16-bit Timers

The M62 implements three, custom, 16-bit timers in onboard, external logic which are capable of triggering processor interrupts and acting as clock sources for I/O modules and external hardware. The timers provide readback capability for the current count register, which allows them to be used as digital event counters. Each channel may be driven by either an onboard 10 MHz clock source or by an external clock input. In addition, external gating signal are available for each timer channel which allow an external TTL signal to selectively enable or disable the timer’s clock input to control counting.

All three timers consist of 16-bit decrementing free-running counters with matching 16-bit period registers. Driven by a source timebase, the timers decrement once per input clock until they reach zero, whereupon they automatically reload from the period register and continue counting down. The timer output is normally high and falls low for one source clock cycle upon expiration of the counter value. The source clock may be selected from either a 10 MHz onboard timebase or an external input pin. The source clock is optionally gateable via a second set of gate inputs. The gating and clock input options allow the external timers to act as event counters for external hardware.

The following table shows the memory map for the timer control registers.

TABLE 8. **External Timer Control Registers**

Function	Address
Clock Mode	0x14F0000
Channel 0 Period	0x14F0008
Channel 1 Period	0x14F000C
Channel 2 Period	0x14F0010
Channel 0 Count	0x14F0014
Channel 1 Count	0x14F0018
Channel 2 Count	0x14F001C

The clock mode register controls the source of the clock used to drive each channel. Data bus bit D0 controls channel zero, D1 controls channel one, and D2 controls channel two. Writing a zero to a bit selects the onboard 10 MHz clock source as the timers clock input, while writing a one selects the external INCLKx inputs available on the digital I/O connector. The INCLKx inputs allow for each of the three channels to be driven by independent clocks from external hardware. For example, writing the hex value 0x2 to the clock mode register selects the 10 MHz clock as the source timebase for channels zero and two, while channel one is driven by the INCLK1 signal.

The timer period registers are used to store the period values for each timer channel. Each timer's output pulse period is equal to the period register value plus two source clock cycles. For example, if the clock mode register for channel zero was programmed to select the 10 MHz clock as the source clock for channel zero, and the period register were programmed with the value 98, then timer channel zero's output pulse would occur with a period of (98 + 2) source clock cycles, or a frequency of

10 MHz/(98 + 2) = 100 kHz

The highest legal value for the period register is 65534 (yielding a lowest possible output frequency of 152 Hz when using the 10 MHz onboard source clock). Please note that the a period register write causes an immediate counter reload (i.e. the counter immediately starts counting down from the new period value).

The timer gate inputs allow external signals to control when the counter will decrement. Pulling the gate line low will disable clocking of the appropriate timer channel. The gate inputs are individually pulled up to 5V via a 10K resistor.

The timer output signals (TMR0, TMR1, and TMR2 for channels zero, one, and two respectively), input clock gating signals (GATE0, GATE1, and GATE2), and input clocks (INCLK0, INCLK1, and INCLK2) are all available for external connections on the digital I/O connector (see the appendices for pinouts).

AD9850 Direct Digital Synthesizer

The AD9850 direct digital synthesizer (DDS) is a precision programmable clock source which is capable of generating frequencies in the range of 0 to 25 MHz with a resolution of 0.019 Hz/step. Unlike a digital counter-timer chip which uses a digital counter to divide down a high input clock rate, the DDS uses phase-locked-loop synthesizer technology to tune a sine wave oscillator based on an 32-bit digital word. This method realizes a linear output frequency over input range rather than the nonlinear one associated with counter-timer chips, whose resolution drops dramatically as the period register used to program them falls. The counter-timer device has a nonlinear frequency step change over its input code range, as opposed to the DDS device which maintains a linear frequency step for each input code increment. This results in the counter-timer's increased resolution at the high end of its input code range, with a correspondingly low resolution at the low end. The AD9850 timebase should be selected for use when a fairly fast but very precise and accurate clock is required by the application.

The AD9850 is mapped into I/O space as shown in the table below. The device is interfaced using the parallel I/O method, with an address to write data, one to trigger frequency/phase updates, and one to control the reset pin of the device.

Function	**I/O Space Address**
AD9850 Reset	0x1470000
AD9850 Frequency Update	0x1480000
AD9850 Write Clock	0x1490000

TABLE 9. **AD9850 Control Registers**

The write clock address latches frequency/phase data into the AD9850 one byte at a time. The least significant eight bits of the processor bus carry the bytewide data. The frequency update address causes the output frequency and phase of the DDS clock to update to the values contained in its input latches. The reset address causes an active high reset pulse to be generated to the AD9850. All three registers are write-only.

The M62 Development Package includes a routine (`timebase()`) which makes it easy to set the AD9850's output frequency (see the *M62 Development Package Software Manual* for details).

Digital I/O

The M62 includes 32 bits of software programmable digital I/O for use in controlling digital instruments or acquiring digital inputs. The digital I/O port controls are mapped into memory space using three addresses: one to read/write the digital I/O data as a single 32-bit word, one for direction control for each byte of the port, and one for controlling the source of the clock edge used to latch input data into the digital I/O port register. The following table lists the addresses and their functions.

Function	**Address**
Digital I/O Data Register	0x14A0000
Digital I/O Direction Control	0x14B0000
Digital I/O Input Latch Clock Control Register	0x14C0000

TABLE 10. **Digital I/O Control Registers**

The direction control register provides for software control of the drive direction of the port. The least significant four bits of the register control the four bytes available on the I/O port. Bit D0 sets the direction for the least significant eight bits if the port (port bits 0-7), D1 the next least significant bits (8-15), D2 the next least significant (16-23) and D3 the most significant (24-31). Each byte is individually controllable by writing a zero (to select output) or a one (to select input) to the respective bit in the direction control register. For example, if the value 0xC were written to the direction control register, bits 0-15 would act as inputs while bits 16-31 would act as outputs. All bytes default to input mode upon board powerup or reset.

The data register allows software to directly read data from port pins programmed for input, or write data to pins programmed for output. Read operations performed from the data register on port bytes programmed for output will return the current value of the digital I/O latch (i.e. the last value written to that portion of the port). For example, suppose that the direction control were programmed to 0xC and the data register written with the data word 0x12340000. Since the most significant 16 bits are setup as outputs, those pins on the port connector would assume the value

0x1234. A subsequent read of the port would yield the value 0x1234xxxx, where xxxx is the value of the signals present on the digital I/O connector.

The input latch clock register allows the user to select either software read clocking or external hardware clocking. Writing a zero to the register selects software clocking, while writing a one selects external hardware clocking. If software clocking is selected, then the port latches programmed for input will clock in the digital data present on the external pins at the beginning of a read cycle executed on the port data register (30-50 ns before the data is returned to the processor, depending on processor clock speed). If external clocking is selected, then the port will latch data on the falling edge of the TTL signal EXT_DIG_RD_CLK* on the digital I/O connector. The data will be held for the processor to read until the next low-going edge of the EXT_DIG_RD_CLK* signal. In the external hardware clocking mode, read operations by the processor do not affect the contents of the digital I/O latch. The latched data may be reread as many times as is required, and only another EXT_DIG_RD_CLK* pulse will cause new data to be latched into the port.

The ‘FCT16952 devices used to implement the digital I/O port are capable of sourcing 32 mA and sinking 64 mA per pin.

Digital I/O Timing

The following diagram gives timing information for the digital I/O port when used in external readback clock mode (see above for details). This data is derived from device specifications and is not factory tested.

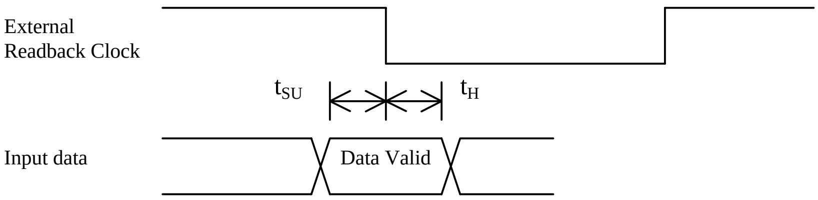


FIGURE 8. Digital I/O Port Timing

Parameter	min (ns)
t_{SU}	0
t_H	10

TABLE 11. **Digital I/O Port Timing Parameters**

External Mux Control

The M62 provides two external multiplexer control bus connectors for use with the TERM line of external multiplexer boards. Control for the multiplexer connectors is provided at the addresses listed in the following table.

TERM Module	Function	I/O Space Address
0	Mux #0 Channel Select	0x14D0000
	Mux #1 Channel Select	0x14D0004
	Mux #2 Channel Select	0x14D0008
	Mux #3 Channel Select	0x14D000C
	All Muxes Channel Select	0x14D0010
	Reset	0x14D001C
1	Mux #0 Channel Select	0x14E0000
	Mux #1 Channel Select	0x14E0004
	Mux #2 Channel Select	0x14E0008
	Mux #3 Channel Select	0x14E000C
	All Muxes Channel Select	0x14E0010
	Reset	0x14E001C

TABLE 12. **TERM Function Memory Map**

The control connectors (JP25 for TERM module 0 and JP26 for TERM module 1) select multiplexer channel numbers. The first four addresses from the start of each mux control address map allow the selection of incoming signals on each multiplexer device on the TERM. The fifth address location allows the simultaneous selection of the same channel on all multiplexer devices. The remaining address performs a global reset of the TERM hardware.

See the OMNIBUS Hardware Manual for additional information regarding the use of Innovative’s TERM modules with the M62.

Interrupts

The ‘C6201 processor implements four interrupt input pins which allow external hardware events to directly trigger software activity. Processor interrupt inputs are supported on the M62 through a set of control registers and multiplexers which allows application software to dynamically select the source of the signal which will drive each particular interrupt input.

The available interrupt source signals are as follows:

1. External interrupt input pins 0-3 (from the I/O modules)
2. External timer channels 0-2
3. 9850 direct digital synthesizer clock
4. PCI bus interrupt
5. Various receive and transmit FIFOPort level status

The following table shows the addresses of the control registers for each processor interrupt input. A value written to the appropriate control register causes the interrupt mux to select the interrupt source given in the next two tablea (see below). Note that the selections vary depending on which interrupt input is being programmed.

Function	Address
External Interrupt Input 4 Select	0x1500000
External Interrupt Input 5 Select	0x1510000
External Interrupt Input 6 Select	0x1520000
External Interrupt Input 7 Select	0x1530000

TABLE 13. **External Interrupt Input Control Registers**

Interrupt Control Register Value	Interrupt Source
0	External Interrupt Input 0
1	External Interrupt Input 1
2	External Interrupt Input 2
3	External Interrupt Input 3
4	External Timer 0
5	External Timer 1
6	External Timer 2
7	9850 DDS Clock

8	PCI bus
9	Receive FIFOPort empty
10	Receive FIFOPort half full
11	Receive FIFOPort full
12	Receive FIFOPort almost-full
15	Deactivated (interrupt held high)

TABLE 14. **Interrupt Source 4 and 5 Select Register Values**

Interrupt Control Register Value	**Interrupt Source**
0	External Interrupt Input 0
1	External Interrupt Input 1
2	External Interrupt Input 2
3	External Interrupt Input 3
4	External Timer 0
5	External Timer 1
6	External Timer 2
7	9850 DDS Clock
8	PCI bus
9	Transmit FIFOPort empty
10	Transmit FIFOPort half full
11	Transmit FIFOPort full
12	Transmit FIFOPort almost-full
15	Deactivated (interrupt held high)

TABLE 15. **Interrupt Source 6 and 7 Select Register Values**

For example, if the application requires the output from external timer channel two to drive processor interrupt input five, the value six should be written to memory location 0x1510000. All interrupt control registers default to setting 15 (disabled) on powerup or board reset. Note that the processor interrupt signals generated by the logic are active low (falling edge trigger), and the ‘C6201 interrupt polarity control register must be programmed to the value 0xF to correctly receive interrupts.

JTAG Test Bus

The M62 implements a JTAG 1149.1-compatible scan path loop through the onboard ‘C6201, with connector compatible with the specification provided in the *TMS320C6201 User’s Guide*. When connecting a JTAG controller card cable (from an Innovative Integration Code Hammer debugger card, Texas Instruments

XDS-510, or other vendor's JTAG hardware), the JP11 connector is used. A shunt should always be installed on jumper JP13 when the JTAG debugger is in use.

Note: the M62 design boots the 'C6201 processor using the HPI boot mode. On device power up or reset, it is not possible to start JTAG debugger software until after the HPI boot process has been completed (the software will return with a "cannot init target" error message if it is started after the processor has been released from reset but before the processor has finished the boot process). Innovative Integration includes a small bootstrapping utility (BOOT.EXE) in the M62 Zuma Toolset which will bootload a small test application onto the M62 and which should be used prior to starting the debugger after a reset.

M62 PCI Bus Features

The M62 uses the V360 PCI bus bridge chip, along with external glue logic and asynchronous SRAM, to implement its interface to the PCI bus. The V360 acts as a bridge chip to translate accesses from the PCI bus into accesses on the 'C6201 bus, and also performs DMA style data transfers between PCI address space and the M62's asynchronous SRAM. Access to the 'C6201's HPI port through the V360 is used by host applications to bootload software into the 'C6201.

PCI Bus I/O and Memory Map

The M62 uses the V360 base address registers and address apertures to map three sets of functionality into PCI bus I/O and memory space: the V360 internal registers, the async SRAM memory, and the processor's host port interface. Address assignments are made to the board via PCI configuration cycles on system powerup, or by the host operating system. The following descriptions of the addressed features assume a working knowledge of PCI plug and play technology as well as any host operating system support provided by the system in use. The M62 Development Package provides host drivers and access support and is highly recommended to shorten software development time.

The V360's internal register set is mapped into I/O space on the PCI bus using base address register zero (PCI_IO_BASE in the V360 data sheet), and allows host access to all of the features of the bridge chip. Host accesses to the I/O space in which the registers are mapped result in slave responses from the V360 device. The lowest part of the register set also provides a convenient access point to the PCI configuration space registers of the device.

The async SRAM is mapped into host PCI memory using base address register one (PCI_BASE0 in the V3 literature). This allows the host processor to gain slave mode access to the SRAM memory on the M62 for data transfers, and also allows other expansion boards to act as bus masters to the M62 for direct data transfers. Accesses to the PCI mapping address by either the host processor or another bus master result in slave responses from the V360. An async SRAM access results in a HOLD/HOLDA arbitration request by the V360 device to the ‘C6201: the slave access will be held not ready until the ‘C6201 has dropped into hold mode and released access to the M62’s processor bus. The V360 then completes the required transfer between the PCI bus and the async SRAM and releases the HOLD request t the ‘C6201.

Please note that the ‘C6201 must be in a state where it is capable of releasing bus ownership to the V360 for the PCI bus access to complete normally. Do not set the NOHOLD bit in the EMIF Global Control Register prior to attempting slave accesses from the PCI bus. Also note that the base address register used to map the async SRAM into PCI bus space requests 32-bit address mapping, which means that 32-bit capable host software is required to access the async SRAM memory.

The ‘C6201’s HPI feature is also mapped into PCI bus memory, using the V360’s base address register 2 (PCI_BASE1 in the V360 data sheet). The following table gives the mapping of the various HPI registers within the PCI bus address space.

PCI Bus BAR1 Offset	Function
0x0	HPIC
0x4	HPIC
0x8	HPIA low half word
0xC	HPIA high half word
0x10	HPID low half word, with address autoincrement
0x14	HPID high half word, with address autoincrement
0x18	HPID low half word, without address autoincrement
0x1C	HPID high half word, without address autoincrement

TABLE 16. **HPI Port PCI Bus Mapping**

Accesses within the PCI mapped HPI interface result in slave responses from the M62. The various registers of the HPI interface are mapped as shown in the above table, and have the read/write limitations noted in the TMS320C62xx Peripherals Reference Guide. The HPI interface allows access to both the standard HPID interface (without address autoincrementing) and to the HPID mapping which causes the current address register to be incremented automatically with each data access.

The HPI port interface uses software ready monitoring to poll the current status of the interface. Host software must poll the status of the HRDY bit in the HPIC register to determine if a current access is finished and a new access may be started.

M62 Bootstrapping

The M62 processor operates in HPI boot mode and supports direct host access to the processor's HPI port via memory mapped registers on the PCI bus. This feature allows the host to access any 'C6201 memory location and is intended for processor bootstrapping.

The 'C6201 boot process involves the following steps:

1. Toggle the processor reset active, then inactive.
2. Via the HPI interface, place a bootstrap compatible code image in the processor's internal memory starting at address zero.
3. Once the code has been placed in processor memory, write a one to the DSPINT bit in the HPIC register to wake the CPU from the reset state. The processor will then begin software execution starting at address zero in internal memory.

Although the HPI MAP1 boot mode begins running software from onchip memory, it is possible to load code anywhere in offchip memory provided that the bus control registers for the memory area in question are initialized prior to any writes via the HPI interface.

There is a complete host COFF compatible M62 bootload routine included in the M62 Development Package which facilitates 'C6201 processor bootloading.

CHAPTER 2 Appendices

Connector pinouts

JP17, JP18, JP21, JP22, P1, P2 - OMNIBUS I/O Connectors (M62 only)

Connector types:	JP17, JP21: AMP .05 Subminiature D male
	JP18, JP22: .100” header
	P1, P2: Male DB15 connector
Number of pins:	JP17, JP21: 50
	JP18, JP22: 50
	P1, P2: 15
Mating connector:	JP17, JP21: AMP 173279-3
	JP18, JP22: AMP 1-746285-0
	P1, P2: AMP 747909-2

The following table shows the interconnections between the JP17 (OMNIBUS slot 0) and JP21 (OMNIBUS slot 1) module I/O connectors and their respective external I/O connectors, JP18 and P1 (OMNIBUS slot 0) and JP22 and P2 (OMNIBUS slot 1).

JP17, JP21 Pin Numbers	**JP18, JP22 Pin Numbers**	**P1, P2 Pin Numbers**
1-35	1-35	NA
36	36	1
37	37	9
38	38	2
39	39	10
40	40	3
41	41	11
42	42	4
43	43	12
44	44	5
45	45	13
46	46	6
47	47	14
48	48	7
49	49	15
50	50	8

TABLE 17. **OMNIBUS I/O Connector Pinouts**

JP17, JP18, JP21, JP22, JP32, JP33 - OMNIBUS I/O Connectors (cM62 only)

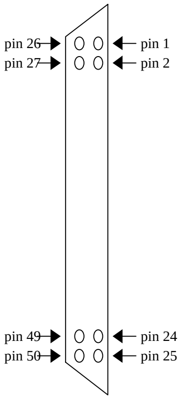
Connector types:	JP17, JP21, JP32: AMP .05 Subminiature D male
	JP18, JP22, JP33: AMP Amplimite Series III
Number of pins:	JP17, JP21, JP32: 50
	JP18, JP22, JP33: 50
Mating connector:	JP17, JP21, JP32: AMP 173279-3
	JP18, JP22, JP33: AMP 750737-5

The following table shows the interconnections between the JP17 (OMNIBUS slot 0), JP21 (OMNIBUS slot 1), and JP22 (OMNIBUS slot 2) I/O connectors and their respective external I/O connectors, JP18 (OMNIBUS slot 0), JP22 (OMNIBUS slot 1), and JP33 (OMNIBUS slot 2).

JP17, JP21, JP32 Pin Numbers	**JP18, JP22, JP33 Pin Numbers**
1-35	1-35
36	36
37	37
38	38
39	39
40	40
41	41
42	42
43	43
44	44
45	45
46	46
47	47
48	48
49	49
50	50

TABLE 18. **OMNIBUS I/O Connector Pinouts**

The following diagram gives the physical pin locations for JP18, JP22, and JP33 connectors on the cM62 board. Please note that these physical pin positions do not use the same numbering scheme as standard SCSI 50 pin connectors.



Not to scale. Front view of I/O connector v
edge of the board towards the top of the dra

FIGURE 9. **OMNIBUS I/O Connector Pin Configuration**

JP19, 20, 23, 24, 34, 35 - OMNIBUS Bus Connectors

Connector types:	AMP .05 Subminiature D male
Number of pins:	50
Mating connector:	AMP 173279-3

The following table gives the pin numbers and functions for the JP19 (OMNIBUS slot 0), JP23 (OMNIBUS slot 1), and JP34 (OMNIBUS slot 2) (available only on the cM62) connectors. The functions for JP23 and JP34 are identical to those of JP19, except where noted.

Pin Number	**JP19 Function**	**JP23 Function**	**JP34 Function (cM62 only)**	**Direction (from M62)**
1, 19	Digital +5V			O, power
2, 20	Digital ground			O, power
3-18	Data bus 0-15			I/O
21, 43, 40, 45, 39, 26, 27	Address bus 2-8			O
28	Reset (active low)			O
29	External interrupt 0	External interrupt 2	External interrupt 4	I
30	Bus ready (active low)			I (open-collector)
31	Processor CLKOUT2 / 4			O
32	PIT timebase channel 0			O
33	R/W*			O
34	9850 timebase			O
35-38	IOMOD0-3 decoded selects (active low)	IOMOD4-7 decoded selects (active low)	IOMOD8-11 decoded selects (active low)	O
25	-12V			O, power
23	+12V			O, power
41,42	Analog ground			O, power
22,24	Analog -15V			O, power
44,46	Analog +15V			O, power
47,49	Analog +5V			O, power
48,50	Analog -5V			O, power

TABLE 19. I/O Module Bus Connectors

The following table gives the pin numbers and functions for the JP20 (OMNIBUS slot 0), JP24 (OMNIBUS slot 1), and JP35 (OMNIBUS slot 2) (available only on cM62) connectors. The functions for JP24 and JP35 are identical to those of JP20, except where noted.

Pin Number	**JP20 Function**	**JP24 Function**	**JP35 Function (cM62 only)**	**Direction (from M62)**
1, 3-6	Address bus 9-13			O
2, 19, 20, 49, 50	Digital ground			O, power
7-18	Reserved	Reserved	Reserved	NA
21	PIT timebase channel 1			O
22	External trigger 0	External trigger 1	External trigger 1	O
23,25	+12V (from PCI bus)			O, power
24	CLKS0	CLKS1	CLKS1	I
26	CLKR0	CLKR1	CLKR1	I/O
27	FSR0	FSR1	FSR1	I/O
28	CLKX0	CLKX1	CLKX1	I/O
29	External interrupt 1	External interrupt 3	External interrupt 5	I
30	DR0	DR1	DR1	I
31	FSX0	FSX1	FSX1	I/O
32	DX0	DX1	DX1	O
33-48	Data bus 16-31			I/O

TABLE 20. I/O Module Bus Connectors

JP14 – Digital I/O Connector

Connector type:	0.100” square double row shrouded header
Number of pins:	50
Mating connector:	AMP 1-746285-0

The following table gives the pin numbers and functions for the JP14 connector.

Pin Number	JP14 Function	Direction (from M62)
1-32	Digital I/O bit 0..31	I/O
33	External Trigger 0 Input (active low)	I
34	9850 DDS Clock Output	O
35	External Trigger 1 Input (active low)	I
36, 38, 40	External Timer Ch. 0, 1, 2 Clock Outputs	O
37, 39, 41	External Timer Ch. 0, 1, 2 Gate Inputs	I
42, 44, 46	External Timer Ch. 0, 1, 2 Timebase Inputs	I
45	External Digital Readback Clock (active low)	I
47	Digital +5V	Power
49	Digital Ground	Power
43, 48, 50	Reserved	NA

TABLE 21. **Digital I/O Connector**

JP31 – Miscellaneous Digital I/O Connector

Connector type:	0.1” square header
Number of pins:	10
Mating connector:	AMP 111811-1

The following table gives the pin numbers and functions for the JP31 connector.

Pin Number	JP31 Function	Direction (from M62)
1	External Trigger 0 Input (active low)	I
2	External Timer 0 Clock Output	O
3	On-chip Timer 1 Out	O
5	On-chip Timer 1 In	I
7	On-chip Timer 0 Out	O
9	On-chip Timer 0 In	I
4,6,8	Reserved	NA
10	Digital Ground	Power

TABLE 22. Miscellaneous Digital I/O Connector

JP15, JP16 – Processor Serial Port Connectors

Connector type:	10 pin shrouded header
Number of pins:	10
Mating connector:	AMP 746285-1 (for ribbon cable termination) or Samtec SSQ style (for board-board applications)

The following table gives the pin numbers and functions for the JP15 and JP16 connectors. Pin functions of JP16 are identical to those of JP15 except where noted.

Pin Number	JP15 Function	JP16 Function	Direction (from M62)
1	CLKS0	CLKS1	I
2	FSR0	FSR1	I/O
3	CLKR0	CLKR1	I/O
4	FSX0	FSX1	I/O
5	CLKX0	CLKX1	I/O
6	Digital 3.3V		Power
7	DR0	DR0	I
8	Digital 5V		Power
9	DX0	DX0	O
10	Digital Ground		Power

TABLE 23. **Processor Serial Port Connector**

JP11 – JTAG Debugger Connector

Connector type:	14 pin shrouded header
Number of pins:	14
Mating connector:	AMP 746285-2

The following table gives the pin numbers and functions for the JP11 connector. This connector follows the recommendations given in section 13 of the *TMS320C62xx Peripherals Reference Guide*.

Pin Number	JP11 Function	Direction (from M62)
1	TMS	I
2	TRST*	I
3	TDI	I
5	Digital +3V	Power
7	TDO	O
9,11	TCK	I
13	EMU0	I/O
14	EMU1	I/O
4, 6, 8, 10, 12	Digital ground	Power

TABLE 24. JTAG Debugger Connector

JP30 – FIFOPort Connector

Connector type:	2mm header
Number of pins:	54
Mating connector:	Samtec SQW style

The following table gives the pin numbers and functions for the JP30 connector.

Pin Number	JP30 Function	Direction (from M62)
1	Digital 5V	Power
2, 44	Ground	Power
3-18	Input Data Bits 15-0	I
19	Reserved	NA
20	Input Strobe	I
21	On-chip Timer 1 Out	O
22	On-chip Timer 1 In	I
23	On-chip Timer 0 Out	O
24	On-chip Timer 0 In	I
25-40	Output Data Bits 0-15	O
41	External Interrupt Input	I
42	Output Strobe	O
43	Digital 3.3V	Power
45	Half-full Flag Out	O
46	Almost-full Flag Out	O
47	Almost-full Level Control In	I
48	Full Flag Out	O
49	Almost-full Level Control Out	O
50	Empty Flag Out	O
51	Half-full Flag In	I
52	Almost-full Flag In	I
53	Empty Flag In	I
54	Full Flag In	I

TABLE 25. **FIFOPort Connector**

TMS320C6201 Limitations and Errata

As of this writing, the TMS320C6201 processor has several limitations and errata which can affect the maximum clock rate at which the processor can successfully run and which may impede the proper operation of certain software applications. This section discusses limitations discovered in Innovative's testing of early M62 prototype cards, as well as errata announced by Texas Instruments regarding the current 2.0 revision silicon.

This information is being supplied to current customers and potential users of the M62 in an effort to keep you informed of the state of 'C6201 processor development and any performance limitations imposed on the M62 hardware design. Innovative will continuously update this information as new data becomes available and particularly when new silicon revisions are released by Texas Instruments to us for testing.

Processor Speed Limitations and External Memory

The current revision 2.0 silicon 'C6201 devices have bus timing issues which prohibit the use of full speed external synchronous burst SRAM (SBSRAM) and synchronous DRAM (SDRAM) devices. These limitations affect the maximum processor speed Innovative will be able to ship with current processors, dependant on the external memory configuration of the M62 hardware as ordered by the customer.

Specifically, Texas Instruments has announced that SDRAM support is limited to approximately 90MHz operation (180 MHz processor speed), while SBSRAM operation is limited to 133 MHz (133 MHz processor speed). These figures were determined through board level testing at Texas Instruments.

Testing of hardware at Innovative Integration shows that SDRAM is supported at least through 80 MHz (160 MHz processor speed), while SBSRAM has been tested up to a rate of 80 MHz (160 MHz processor speed with SBSRAM in half-rate mode). Testing at a processor speed of 133 MHz with SBSRAMs running in full rate mode has shown a read/write failure rate of about 10 ppm. Texas Instruments has specified that not all SBSRAMs are reliable in their tests, so it is possible that the devices Innovative is currently using may not be up to spec. Innovative is in the process of procuring SBSRAM samples of the manufacturer and type used by Texas Instruments and will continue testing of the external memory interface to determine of certain devices are more reliable. In addition, Innovative will be procuring additional clock source devices which will allow the testing of intermediate processor speeds (180 MHz, for example).

Current processors are capable of 200 MHz operation and have been tested at this rate on the M62 design. These tests involve running software strictly from on-chip memory. The external peripheral interfaces (I/O bus sites, serial ports, FIFOPorts, onboard peripherals, async SRAM, PCI interface) are unaffected by the memory interface issue as they use less aggressive bus timing.

Innovative Integration’s policy on processor speed is to deliver the fastest possible speed consistent with the requested external memory configuration on the board. In situations where customers order external memory, Innovative must downgrade the processor speed to match the memory interface limitations. Current processor speeds available versus memory requirements are as follows:

External Synchronous Memory Type	**Delivered Processor Speed**
None	200 MHz
SDRAM	160 MHz
SBSRAM	160 MHz (SBSRAM operating in half-rate mode)

As Innovative continues testing the memory subsystems of the M62, these rates may change to improve the memory access and processor speeds.

Texas Instruments Device Errata

The current Texas Instruments device errata for revision 2.0 silicon TMS32C6201 devices is attached below. At this time Innovative Integration does not consider these errata to be significant to the overall operation of the card design.

TMS320C6201 SILICON ERRATA

The following is a list of problems on TMS320C6201 2.0 silicon or any lower revision. TI creates a new document revision when a new silicon bug is discovered. However, TI does NOT update previously edited files. For example, if you have silicon revision 1.0 and the latest silicon revision is 2.0, you should look at the latest silicon errata for 2.0, as it will also contain any problems found in silicon version 1.0.

Note:

- New items in this document are
  - All revision 2.0 silicon problems.
- All revision 2.0 silicon problems except the below are fixed in 2.1 silicon (test cases validated in simulation).
  - Problem 2.0.6 will be fixed in revision 3.0.
- All revision 1.X silicon problems are fixed in revision 2.0. Silicon revision 1.0 problems 1.0.12 and 1.0.23 were documentation errors, not silicon errors. The documentation has been fixed.
- Problems 1.0.1 and 1.0.4 were fixed in revision silicon revision 1.1.
- Silicon revision is identified by a code in the lower left-hand corner of the chip. The code is of the format Cxx-yyww. If xx is 00 then the silicon is revision 1.0. If xx is 20 then the silicon is revision 2.0.

List Of Bugs

REVISION 2.0 SILICON BUGS

Problem 2.0.1 Program Fetch: Cache Modes Not Functional

WORK-AROUND: Use internal program memory in mapped mode.

Problem 2.0.2 Bootload: Boot from 16-bit and 32-bit Asynchronous ROMs Not Functional

16-bit wide ROM mode works in run time without bugs. The problem is only in boot. . Internal Reference Number 3088.

WORK-AROUND: Place all code in the lowest byte of the boot ROM.

Problem 2.0.3 DMA Channel 0 Split Mode Combined with Auto-initialization Performs Improper Re-Initialization

The source address (transmit read address) is reset one cycle too early when both split mode and auto-initialization are enabled. The bug exists on DMA channel 0 only. Internal Reference Number 3481.

WORK-AROUND: Substitute one of the other channels for channel 0 when this configuration is desired.

Problem 2.0.4 DMA/Program Fetch: Cannot DMA into Program Memory when Running Program From External

Performing a DMA transfer into program memory while running from off-chip can cause invalid program data to read by the CPU. Internal Reference Number 2978.

WORK-AROUND: DMA into program memory only when running from internal program memory.

Problem 2.0.5 Data Access: Parallel Read and Write Accesses to Same EMIF or Internal Peripheral Bus Location Sequenced Wrong

This bug occurs under the following conditions:

- A load and store are in the same execute packet. And Either
  - The addresses both point to off-chip memory through the EMIF, and the load has a destination register in side A (thus the store would have a source register in side B). Or
  - The addresses both point to the peripheral bus, and the load has a destination register in side B (thus the store would have a source register in side A).

When these conditions occur, the store occurs first rather than the load. In general, this will only cause an error if both the load and store addresses are the same. This bug DOES NOT occur if both accesses are to internal data memory. Internal Reference Number 3087.

WORK-AROUND: Avoid loading and storing the same address on the same cycle.

Problem 2.0.6 EMIF: CE Space Crossing on Continuous Request Not Allowed

Any continuous request of the EMIF cannot cross CE address space boundaries. This condition can result in bad data read, or writing to the wrong CE. Internal Reference Numbers 2600 & 3421.

WORK-AROUNDS:

CPU Program Fetch: The simplest fix is for all external program to reside with in a single CE space. Alternatively, program fetch flow should not occur across CE spaces. This can be accomplished by branching on chip in between executing from on CE to another CE.

DMA: All DMA block transfers without read or write synchronization should have all EMIF addresses with in a frame. DMA transfers with read and/or write synchronization as well as CE boundaries crossed between frames are not be affected by this bug.

CPU Data Access: External CPU data accesses cannot perform continuous requests and thus are not affected by this bug.

Problem 2.0.7 EMIF: Reserved Fields Have Incorrect Values

Fields in Bits 15:14 of EMIF CE Space control registers are writeable. They should be read only and have a 0 value. Bits 5:4 of EMIF SDRAM control register are 11b rather than 0. Internal Reference Number s 3248, 3283.

WORK AROUND: Mask these values if 0’s are expected and to only write 0’s to reserved fields.

Problem 2.0.8 EMIF: SDRAM Refresh/DCAB Not Performed Prior to HOLD Request Being Granted

SDRAM is left in the current state when an external HOLD is granted. SDRAM refresh/DCAB is necessary if an interface to a shared memory external SDRAM controller is desired. Internal Reference Number 3249.

WORK-AROUND: Make sure the external controller performs a refresh/DCAB before performing SDRAM accesses

Problem 2.0.9 McBSP New Block Interrupt does not occur for Start of Block 0

When end-of-block interrupt is selected ((R/X)INTM=01b), does not occur at end of frame (i.e. before block 0).

WORK-AROUND: This interrupt is used when on-the-fly channel selection/enabling is being performed. A static channel selection/enabling avoids this.

Problem 2.0.10 (R/X)CBLK Bits in MCR Updated Incorrectly

WORK-AROUND: None. These are status bits and don’t affect McBSP operation.

Problem 2.0.12 McBSP: FRST Improved in 2.1 over 2.0

The following enhancements were made in 2.1.

/FRST=1 is valid only when /GRST=1. In other words the user has to set /FRST=1 only after /GRST=1. If not, write to /FRST=1 is ignored or rather a zero is forced on /FRST by the logic.

During normal operation, when /FRST=1 and /GRST=1, and now the user puts the sample rate generator in reset (/GRST=0) without first clearing the /FRST bit to zero, then the logic will force a zero to /FRST bit before shutting down the sample rate generator.

When /FRST transitions to a 1, the first frame sync is generated after 8 CLKG clocks. The 2.0 implementation was such that the first frame sync was generated after FPER+1 number of CLKG clocks.

Problem 2.0.13 McBSP: /XEMPTY stays low when DXR Written Late

/XEMPTY goes low and stays low when DXR was written on either the last bit or next to last bit of the previous word being transferred to DX. Internal Reference Number 3383.

REVISION 1.0 BUGS

Problem 1.0.1 un-requested SBSRAM write generated

This problem only occurs after a block of SBSRAM reads is immediately followed by a write to on-chip data memory. This problem is limited to EMIF requests generated by the CPU and it does not affect requests originated by the DMA. EMIF requests to other types of memory than SBSRAM are also not affected.

WORK-AROUND: Insert a non-parallel NOP after any block read from SBSRAM that is immediately followed by a write to internal data memory.

Problem 1.0.2 Incorrect switch from SBSRAM and ASRAM cycles

Some data may be lost during switch from SBSRAM to ASRAM cycles. This happens during loads followed by stores, stores followed by loads and loads followed by loads. SBSRAM loads or stores can be sequential or non-sequential.

This problem does not affect other combinations of external memory accesses.

WORK-AROUND: insert at least two non-parallel NOPs between SBSRAM loads/stores and ASRAM loads/stores that immediately follow.

Problem 1.0.3 Cache fetching from off-chip followed by on-chip fetches

This problem occurs only if the second to last fetch packet from off-chip is not fully parallel (contains more than one execute packets). This condition can cause the processor to execute a corrupted execute packet with unpredictable results.

This problem does not occur during all external or all internal fetches. On-chip fetches followed by off-chip fetches are also O.K.

WORK-AROUND: Write code so that the second to last fetch packet fetched from off chip, preceding the on-chip fetches, contains more than one execute packets.

Problem 1.0.4 Cache mode not working

The fetch packets are still fetched from external memory when the cache mode is set. This problem does not result in incorrect execution, just slower operation.

Problem 1.0.5 Wrong address when reads cross from CE0 to CE1

When the external CPU read sequence crosses from CE0 to CE1 space, the first CE1 address in incorrect (same as the last CE0 address).

This problem is not caused by DMA accesses or program fetches.

WORK-AROUND: Do not cross the CE0/CE1 boundary during sequential reads or insert a non-parallel NOP at the boundary.

Problem 1.0.6 Write to ASRAM missed following SBSRAM writes

This problem only occurs when a sequence of writes crosses the memory type boundary.

WORK-AROUND: Insert a non-parallel NOP between sequential writes to different memory types.

Problem 1.0.7 Incorrect SDRAM timing

This problem is caused by a skew of CLKOUT2 relative to all of the SDRAM control signals. The SDRAM will not operate properly without a fix involving external components.

WORK-AROUND: Below is description of what is being done on the TI test card to overcome the CLKOUT2 skew. The jumpers will allow the same circuit to still be used with the upcoming revisions of the C6201 that will have this problem corrected.

```
                                          ^^
                              JP2  ====                    _______
                          _________o  o______             |SDRAM  |
 _____________           |                   |            |       |
|             |          |  |\74LVT240       |            |       |
|   C6201     |__clk02___|__| \o_____________|____sdclk___|>      |
|     A       |             | /UX2                        |       |
 -------------              |/o                            -------
    UX1                       |
                        1  2  |
                vcc o---o  o---
                        3  4  |
                gnd o---o  o---
                    JP1 ====
                         ^^
UX1: DSP
UX2: Inverting Buffer (delays: min=1 ns, nom=2.9ns, and max=4.1ns)
JP1: Jumper to enable/disable the buffer;
JP2: Jumper to bypass the buffer;
NOTES: (power to board is off during configuration changes)
  1. To use the C6201 clk02 directly ... JP1 on  1-2, off 3-4; JP2 on.
  2. To use the inverted C6201 clk02 ... JP1 off 1-2,  on 3-4; JP2 off.
```

Problem 1.0.8 Some signals not reaching full VOH

Affected signal terminals - EMU0, EMU1, TDO, INUMx, and IACK.

The above signals will reach the highest voltage of 1.8V at worst case process and fastest clock speed. At more nominal process they will be above 2V.

No other signals are affected besides the ones listed above. INUMx and IACK signals eventually reach VOH since they don’t switch on every cycle.

WORK-AROUND: At slower speed (CLKOUT1 > 10ns) the affected signals will reach full voltage rails.

Problem 1.0.9 Problem starting the Emulator

When bringing up the emulator, the C6201 must be held at reset until the emulator comes up. This condition only occurs the first time that the Emulator is started following power-up.

Problem 1.0.10 Debugger times-out while single-stepping

This problem can occur while the debugger is single-stepping the external memory. After the time-out error message the debugger may still work in some cases. This condition is rare and it only affects off-chip memory access.

WORK-AROUND: do not single-step through the external memory

Problem 1.0.11 Analysis events not recognized while single-stepping

Problem 1.0.12 EMU0 and EMU1 terminals need external pullups

Use 4.7kOhm external pull-ups with the EMU0 and EMU1 pins. The emulator will not work correctly without the pull-ups.

The future silicon revisions will have internal pull-ups on those signals.

WORK-AROUND: This problem has been corrected on the revision 1 test board, but customers designing their own board should add pullups.

Problem 1.0.13 On-chip program RAM may become corrupted during scan

This problem only occurs when the emulator is scanning data to/from the C6201 operating at speed less then 80MHz. This problem does not occur when the scan chain is not being accessed, or when the C6201 is operating at speeds faster then 80Mhz.

WORK-AROUND: Run the C6201 at CLKOUT1 frequency of 80MHz or above when debugging with the emulator.

Problem 1.0.14 Data not stored for sequential stores involving CPU sides A and B.

Serial stores involving both sides of the CPU (A and B) to SBSRAM and ASRAM may not occur. Memory still holds the old data.

WORK-AROUND: Insert a NOP or another instruction between the two stores.

Problem 1.0.15 Power-down mode doesn’t work with clock mode x1.

Pd3 (formerly idle mode 3) doesn’t shut off the internal clocks when the clock mode is set to x1. Pd3 still works for x2 and x4 clock modes.

WORK-AROUND: Do not use the x1 clock mode.

Problem 1.0.16 Internal program memory reads are failing when using the debugger.

This is caused by Problem 1.0.21.

Problem 1.0.17 INTA and INUM pins may randomly toggle during Emulation

WORK-AROUND: Do not use INTA/INUM pins, or do not use the emulator.

Problem 1.0.18 SDRAM bank select does not maintain valid bank value

The SDRAM bank select is generated on ED09 and for 1M x n x 2 banks also on EA2.0. EA11 does not maintain a valid bank value during all operations, and this causes part of the memory to be inaccessible.

WORK-AROUND: Use 1 x n x 2 type of SDRAM with the bank select of the SDRAM connected to EA18. This will provide valid bank selection for all SDRAM bus cycles.

Problem 1.0.19 External memory reads fail when preceded by an external write

This is probably a DMEMC problem since it affects all memory types.

WORK-AROUND: Prevent back-to-back store/load by inserting a NOP between the store and the load.

Problem 1.0.20 CPU external program fetches corrupt CPU internal data reads.

The corrupted DMEMC reads can also be triggered by DMA accessing the program memory.

WORK-AROUND: Keep the program internal and prevent the DMA from accessing internal program memory when the CPU is accessing the internal data memory.

Problem 1.0.21 Corrupted code when parallel fetch packets cross from external to internal memory.

This problem occurs when using Memory Map 0.

WORK-AROUND: Don't use parallel code when crossing from internal to external program memory.

Problem 1.0.22 Missed SDRAM refresh cycles

Some refresh cycles may be missed if refresh is scheduled to occur during SDRAM read cycles in progress.

NOTICE: This Problem made the revision 1 SDRAM interface unreliable. In addition to fixing the revision 1 bugs, we have decided to completely redesign the SDRAM interface adding full burst functionality and other features. Our recommendation for revision 1 customers at this time is to not use the current revision 1 silicon to interface with SDRAM until the revision 2 silicon is released with the improved SDRAM interface.

Problem 1.0.23 PLL needs different PLLFREQ pin settings

There is a mismatch between bit settings on pins PLLFREQ(3-1) and the CLOCKOUT1 frequency ranges as specified in the Data Sheet. The new values are PLLFREQ(3-1)000=25-135MHz, 001=35-160MHz, 010=40-200MHz.

PLLFREQ(3-1) values 011 and 100 are not valid. If the desired frequency falls inside more then one specified frequency ranges, chose the 000 over 001 and choose the 001 over 010.

Problem 1.0.24 Emulator may not always come up

The correct state of EMU0 and EMU1 may not be latched correctly when bringing up the emulator resulting in emulator "timing out".

This is a very infrequent occurrence that normally doesn't happen unless some special emulation instructions are executed outside of normal boot procedures.

WORK-ROUND: Do system reset and try again

Problem 1.0.25 IACK pin not tested

The timing of the IACK pin is not tested for revision 1 of silicon.

Device Data Sheets

ANALOG DEVICES

CMOS, 125 MHz Complete DDS Synthesizer

AD9850

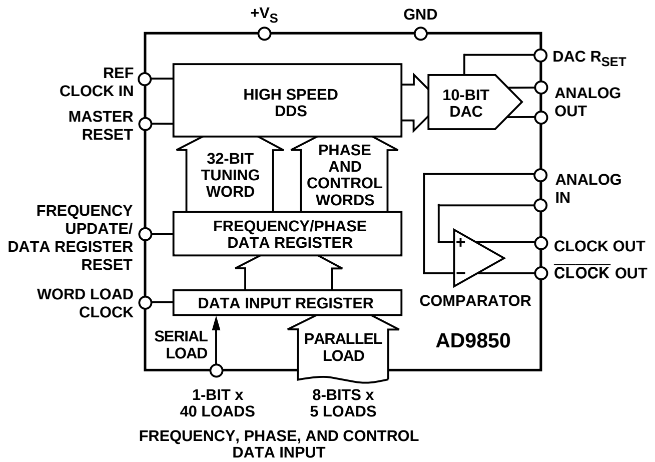
FEATURES

- 125 MHz Clock Rate
- On-Chip High Performance DAC & High Speed Comparator
- DAC SFDR > 50 dB @ 40 MHz AOUT
- 32-Bit Frequency Tuning Word
- Simplified Control Interface: Parallel Byte or Serial Loading Format
- Phase Modulation Capability
- +3.3 V or +5 V Single Supply Operation
- Low Power: 380 mW @ 125 MHz (+5 V)
  155 mW @ 110 MHz (+3.3 V)
- Power-Down Function
- Ultrasmall 28-Lead SSOP Packaging

APPLICATIONS

- Frequency/Phase–Agile Sine-Wave Synthesis
- Clock Recovery and Locking Circuitry for Digital Communications
- Digitally Controlled ADC Encode Generator
- Agile Local Oscillator Applications

FUNCTIONAL BLOCK DIAGRAM



GENERAL DESCRIPTION

The AD9850 is a highly integrated device that uses advanced DDS technology coupled with an internal high speed, high performance, D/A converter and comparator, to form a complete digitally programmable frequency synthesizer and clock generator function. When referenced to an accurate clock source, the AD9850 generates a spectrally pure, frequency/phase-programmable, analog output sine wave. This sine wave can be used directly as a frequency source or converted to a square wave for agile-clock generator applications. The AD9850's innovative high speed DDS core provides a 32-bit frequency tuning word, which results in an output tuning resolution of 0.0291 Hz, for a 125 MHz reference clock input. The AD9850's circuit architecture allows the generation of output frequencies of up to one-half the reference clock frequency (or 62.5 MHz), and the output frequency can be digitally changed (asynchronously) at a rate of up to 23 million new frequencies per second. The device also provides five bits of digitally controlled phase modulation, which enables phase shifting of its output in increments of 180°, 90°, 45°, 22.5°, 11.25° and any combination thereof. The AD9850 also contains a high speed comparator that can be configured to accept the (externally) filtered output of the DAC to generate a low jitter square wave output. This facilitates the device's use as an agile clock generator function.

The frequency tuning, control, and phase modulation words are loaded into the AD9850 via a parallel byte or serial loading format. The parallel load format consists of five iterative loads of an 8-bit control word (byte). The first byte controls phase modulation, power-down enable, and loading format; bytes 2–5 comprise the 32-bit frequency tuning word. Serial loading is accomplished via a 40-bit serial data stream on a single pin. The AD9850 Complete-DDS uses advanced CMOS technology to provide this breakthrough level of functionality and performance on just 155 mW of power dissipation (+3.3 V supply).

The AD9850 is available in a space saving 28-lead SSOP, surface mount package. It is specified to operate over the extended industrial temperature range of –40°C to +85°C.

REV. A

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 617/329-4700 World Wide Web Site: http://www.analog.com
Fax: 617/326-8703

AD9850—SPECIFICATIONS (V_S = +5 V ± 5% except as noted, R_{SET} = 3.9 kΩ)

Parameter	Temp	Test Level	AD9850BRS			Units
			Min	Typ	Max	
CLOCK INPUT CHARACTERISTICS						
Frequency Range						
+5 V Supply	FULL	IV	1		125	MHz
+3.3 V Supply	FULL	IV	1		110	MHz
Pulse Width High/Low						
+5 V Supply	+25°C	IV	3.2			ns
+3.3 V Supply	+25°C	IV	4.1			ns
DAC OUTPUT CHARACTERISTICS						
Full-Scale Output Current						
R_{SET} = 3.9 kΩ	+25°C	V		10.24		mA
R_{SET} = 1.95 kΩ	+25°C	V		20.48		mA
Gain Error	+25°C	I	–10		+10	% FS
Gain Temperature Coefficient	FULL	V		150		ppm/°C
Output Offset	+25°C	I			10	µA
Output Offset Temperature Coefficient	FULL	V		50		nA/°C
Differential Nonlinearity	+25°C	I		0.5	0.75	LSB
Integral Nonlinearity	+25°C	I		0.5	1	LSB
Output Slew Rate (50 Ω, 2 pF Load)	+25°C	V		400		V/µs
Output Impedance	+25°C	IV	50	120		kΩ
Output Capacitance	+25°C	IV			8	pF
Voltage Compliance	+25°C	I			1.5	V
Spurious-Free Dynamic Range (SFDR):						
Wideband (Nyquist Bandwidth)						
1 MHz Analog Out	+25°C	IV	63	72		dBc
20 MHz Analog Out	+25°C	IV	50	58		dBc
40 MHz Analog Out	+25°C	IV	46	54		dBc
Narrowband						
40.13579 MHz ± 50 kHz	+25°C	IV		80		dBc
40.13579 MHz ± 200 kHz	+25°C	IV		77		dBc
4.513579 MHz ± 50 kHz/20.5 MHz CLK	+25°C	IV		84		dBc
4.513579 MHz ± 200 kHz/20.5 MHz CLK	+25°C	IV		84		dBc
COMPARATOR INPUT CHARACTERISTICS						
Input Capacitance	+25°C	V		3		pF
Input Resistance	+25°C	IV	500			kΩ
Input Current	+25°C	I	–12		+12	µA
Input Voltage Range	+25°C	IV	0		V_{DD}	V
Comparator Offset*	FULL	VI	30		30	mV
COMPARATOR OUTPUT CHARACTERISTICS						
Logic "1" Voltage +5 V Supply	FULL	VI	+4.8			V
Logic "1" Voltage +3.3 V Supply	FULL	VI	+3.1			V
Logic "0" Voltage	FULL	VI			+0.4	V
Propagation Delay, +5 V Supply (15 pF Load)	+25°C	V		5.5		ns
Propagation Delay, +3.3 V Supply (15 pF Load)	+25°C	V		7		ns
Rise/Fall Time, +5 V Supply (15 pF Load)	+25°C	V		3		ns
Rise/Fall Time, +3.3 V Supply (15 pF Load)	+25°C	V		3.5		ns
Output Jitter (p-p)	+25°C	V		80		ps
CLOCK OUTPUT CHARACTERISTICS						
Clock Output Duty Cycle (Clk Gen. Config.)	+25°C	IV		50 ± 10		%

Parameter	Temp	Test Level	AD9850BRS Min	 Typ	 Max	Units
CMOS LOGIC INPUTS (Including CLKIN)						
Logic “1” Voltage, +5 V Supply	+25°C	I	3.5			V
Logic “1” Voltage, +3.3 V Supply	+25°C	I	3.0			V
Logic “0” Voltage	+25°C	I			0.4	V
Logic “1” Current	+25°C	I			12	μA
Logic “0” Current	+25°C	I			12	μA
Input Capacitance	+25°C	V		3		pF
POWER SUPPLY (AOUT = 1/3 CLKIN)						
$+V_S$ Current @:						
62.5 MHz Clock, +3.3 V Supply	FULL	VI		30	48	mA
110 MHz Clock, +3.3 V Supply	FULL	VI		47	60	mA
62.5 MHz Clock, +5 V Supply	FULL	VI		44	64	mA
125 MHz Clock, +5 V Supply	FULL	VI		76	96	mA
P_{DISS} @:						
62.5 MHz Clock, +3.3 V Supply	FULL	VI		100	160	mW
110 MHz Clock, +3.3 V Supply	FULL	VI		155	200	mW
62.5 MHz Clock, +5 V Supply	FULL	VI		220	320	mW
125 MHz Clock, +5 V Supply	FULL	VI		380	480	mW
P_{DISS} Power-Down Mode						
+5 V Supply	FULL	V		30		mW
+3.3 V Supply	FULL	V		10		mW

NOTES
*Tested by measuring output duty cycle variation.
Specifications subject to change without notice.

TIMING CHARACTERISTICS* (V_S = +5 V ± 5% except as noted, R_{SET} = 3.9 kΩ)

Parameter	Temp	Test Level	AD9850BRS Min	 Typ	 Max	Units
t_{DS} (Data Setup Time)	FULL	IV	3.5			ns
t_{DH} (Data Hold Time)	FULL	IV	3.5			ns
t_{WH} (W_CLK min. Pulse Width High)	FULL	IV	3.5			ns
t_{WL} (W_CLK min. Pulse Width Low)	FULL	IV	3.5			ns
t_{WD} (W_CLK Delay After FQ_UD)	FULL	IV	7.0			ns
t_{CD} (CLKIN Delay After FQ_UD)	FULL	IV	3.5			ns
t_{FH} (FQ_UD High)	FULL	IV	7.0			ns
t_{FL} (FQ_UD Low)	FULL	IV	7.0			ns
t_{CF} (Output Latency from FQ_UD)						
Frequency Change	FULL	IV	18			CLKIN Cycles
Phase Change	FULL	IV	13			CLKIN Cycles
t_{FD} (FQ_UD Min. Delay After W_CLK)	FULL	IV	7.0			ns
t_{RH} (CLKIN Delay After RESET Rising Edge)	FULL	IV	3.5			ns
t_{RL} (RESET Falling Edge After CLKIN)	FULL	IV	3.5			ns
t_{RS} (Minimum RESET Width)	FULL	IV	5			CLKIN Cycles
t_{OL} (RESET Output Latency)	FULL	IV	13			CLKIN Cycles
t_{RR} (Recovery from RESET)	FULL	IV	2			CLKIN Cycles
Wake-Up Time from Power-Down Mode	+25°C	V		5		μs

NOTES
*Control functions are asynchronous with CLKIN.
Specifications subject to change without notice.

ABSOLUTE MAXIMUM RATINGS*

Maximum Junction Temperature	+165°C
VDD	+6 V
Digital Inputs	–0.7 V to $+V_S$
Digital Output Continuous Current	5 mA
DAC Output Current	30 mA
Storage Temperature	–65°C to +150°C
Operating Temperature	–40°C to +85°C
Lead Temperature (Soldering 10 sec)	+300°C
SSOP θ_{JA} Thermal Impedance	82°C/W

*Absolute maximum ratings are limiting values, to be applied individually, and beyond which the serviceability of the circuit may be impaired. Functional operability under any of these conditions is not necessarily implied. Exposure of absolute maximum rating conditions for extended periods of time may affect device reliability.

EXPLANATION OF TEST LEVELS

Test Level

I – 100% Production Tested.

III – Sample Tested Only.

IV – Parameter is guaranteed by design and characterization testing.

V – Parameter is a typical value only.

VI – All devices are 100% production tested at +25°C. 100% production tested at temperature extremes for military temperature devices; guaranteed by design and characterization testing for industrial devices.

CAUTION

ESD (electrostatic discharge) sensitive device. Electrostatic charges as high as 4000 V readily accumulate on the human body and test equipment and can discharge without detection. Although the AD9850 features proprietary ESD protection circuitry, permanent damage may occur on devices subjected to high energy electrostatic discharges. Therefore, proper ESD precautions are recommended to avoid performance degradation or loss of functionality.

ORDERING GUIDE

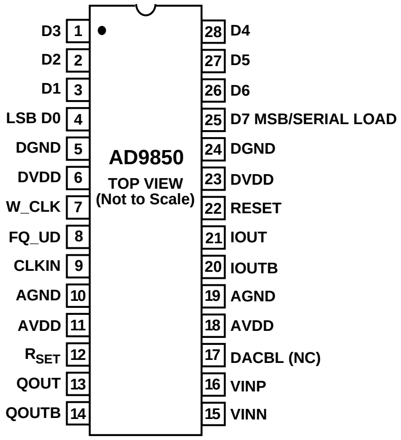
Model	Temperature Range	Package Option*
AD9850BRS	–40°C to +85°C	RS-28

*RS = Shrink Small Outline (SSOP).

Table I. Lead Function Descriptions

Mnemonic	Lead No.	Function
CLKIN	9	Reference Clock Input. This may be a continuous CMOS-level pulse train or sine input biased at 1/2 V supply. The rising edge of this clock initiates operation.
R_{SET}	12	This is the DAC's external R_{SET} connection. This resistor value sets the DAC full-scale output current. For normal applications (F_S I_{OUT} = 10 mA), the value for R_{SET} is 3.9 kΩ connected to ground. The R_{SET}/I_{OUT} relationship is: I_{OUT} = 32 (1.248 V/R_{SET}).
AGND	10, 19	Analog Ground. These leads are the ground return for the analog circuitry (DAC and comparator).
DGND	5, 24	Digital Ground. These are the ground return leads for the digital circuitry.
DVDD	6, 23	Supply Voltage Leads for digital circuitry.
AVDD	11, 18	Supply Voltage for the analog circuitry (DAC and comparator).
W_CLK	7	Word Load Clock. This clock is used to load the parallel or serial frequency/phase/control words.
FQ_UD	8	Frequency Update. On the rising edge of this clock, the DDS will update to the frequency (or phase) loaded in the data input register, it then resets the pointer to Word 0.
D0–D7	1–4, 25–28	8-Bit Data Input. This is the 8-bit data port for iteratively loading the 32-bit frequency and 8-bit phase/control word. D7 = MSB; D0 = LSB. D7 (Pin 25) also serves as the input pin for the 40-bit serial data word.
RESET	22	Reset. This is the master reset function; when set high it clears all registers (except the input register) and the DAC output will go to Cosine 0 after additional clock cycles—see Figure 19.
IOUT	21	Analog Current Output of the DAC.
IOUTB	20	The Complementary Analog Output of the DAC.
DACBL	17	DAC Baseline. This is the DAC baseline voltage reference; this lead is internally bypassed and should normally be considered a "no connect" for optimum performance.
VINP	16	Noninverting voltage input. This is the comparator's positive input.
VINN	15	Inverting Voltage Input. This is the comparator's negative input.
QOUTB	14	Output Complement. This is the comparator's complement output.
QOUT	13	Output True. This is the comparator's true output.

PIN CONFIGURATIONS



NC = NO CONNECT

AD9850–Typical Performance Characteristics

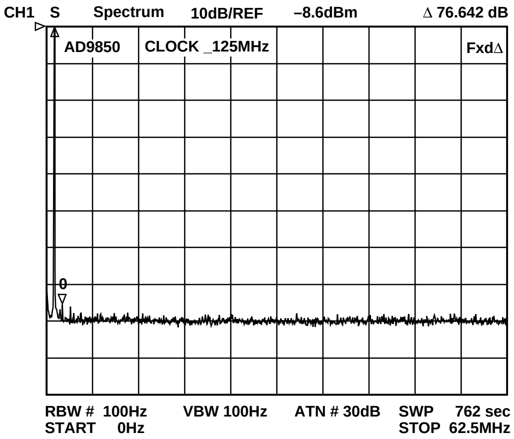


Figure 1. SFDR, CLKIN = 125 MHz/FOUT = 1 MHz

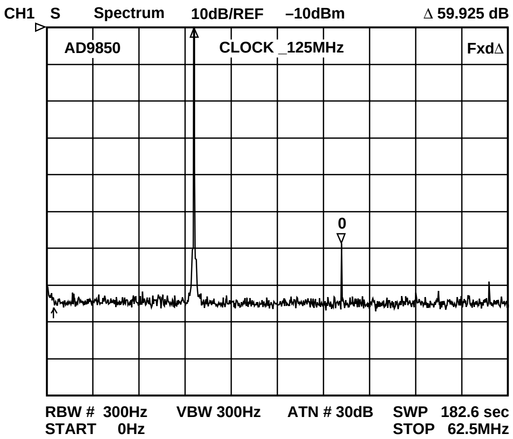


Figure 4. SFDR, CLKIN = 125 MHz/FOUT = 20 MHz

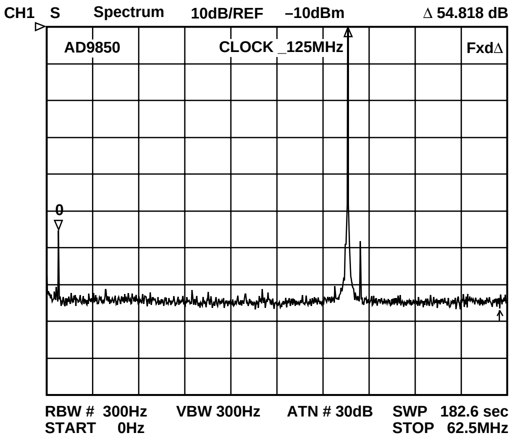


Figure 2. SFDR, CLKIN = 125 MHz/FOUT = 41 MHz

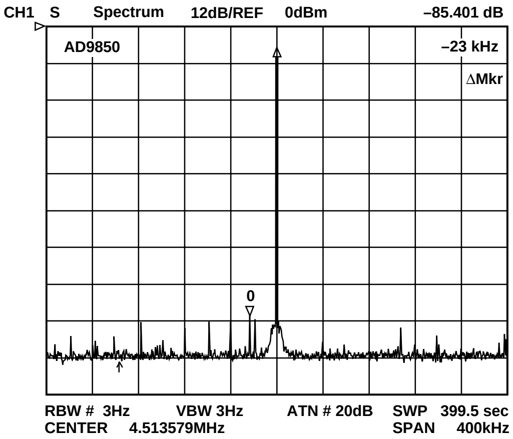


Figure 5. SFDR, CLKIN = 20.5 MHz/FOUT = 4.5 MHz

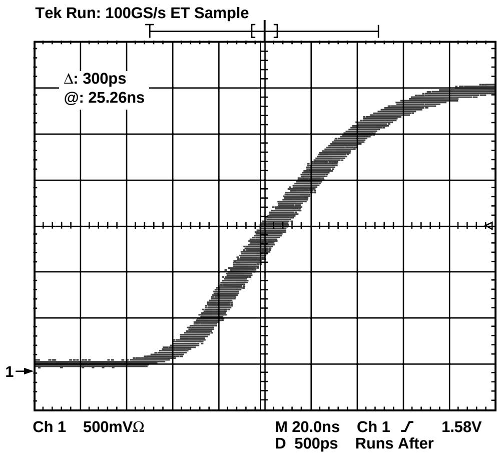


Figure 3. Typical Comparator Output Jitter, AD9850 Configured as Clock Generator w/42 MHz LP Filter (40 MHz AOUT/125 MHz CLKIN)

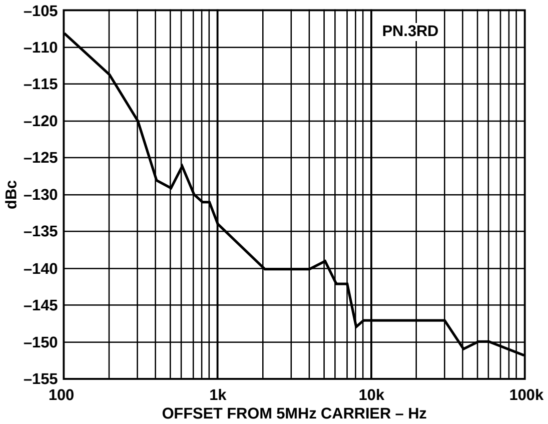


Figure 6. Output Residual Phase Noise (5 MHz AOUT/ 125 MHz CLKIN)

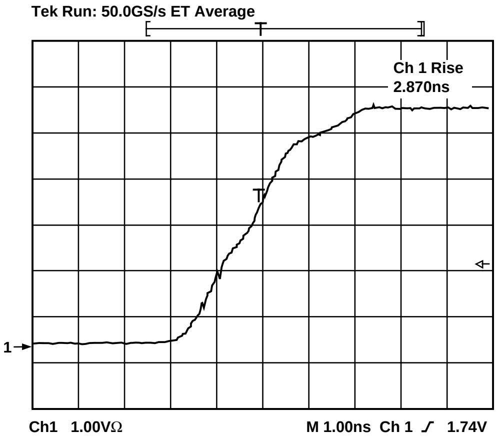


Figure 7. Comparator Output Rise Time (5 V Supply/15 pF Load)

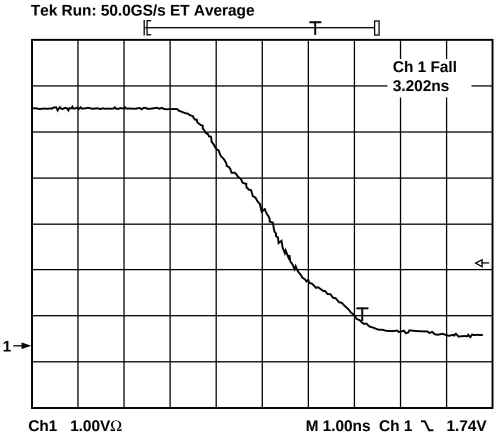


Figure 10. Comparator Output Fall Time (5 V Supply/15 pF Load)

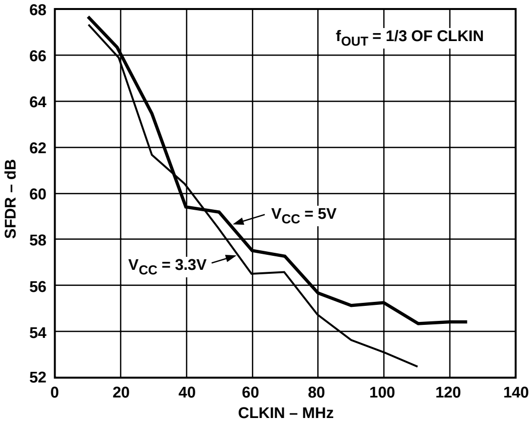


Figure 8. SFDR vs. CLKIN Frequency (AOUT = 1/3 of CLKIN)

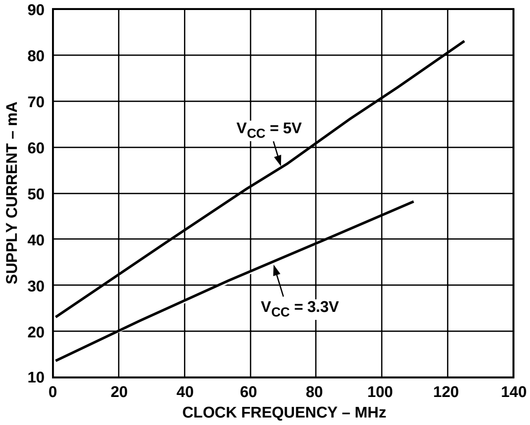


Figure 11. Supply Current vs. CLKIN Frequency (AOUT = 1/3 of CLKIN)

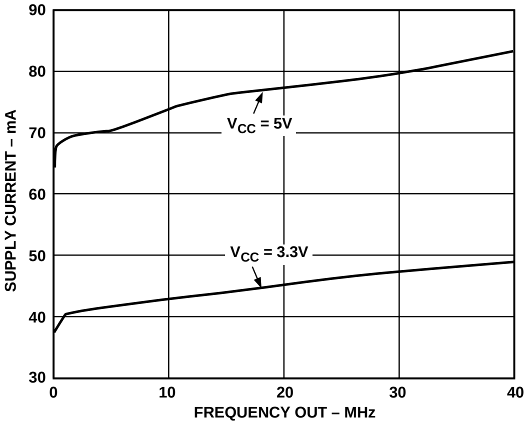


Figure 9. Supply Current vs. AOUT Frequency (CLKIN = 125/110 MHz for 5 V/3.3 V Plot)

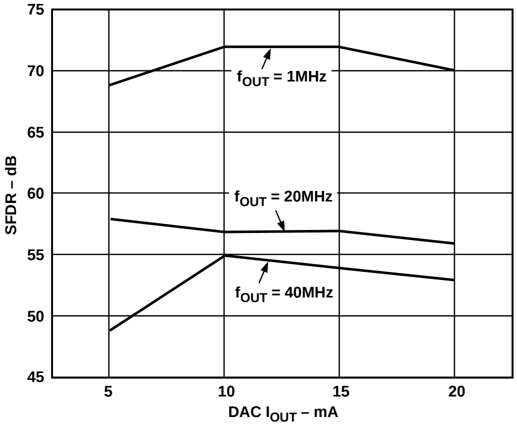


Figure 12. SFDR vs. DAC IOUT (AOUT = 1/3 of CLKIN)

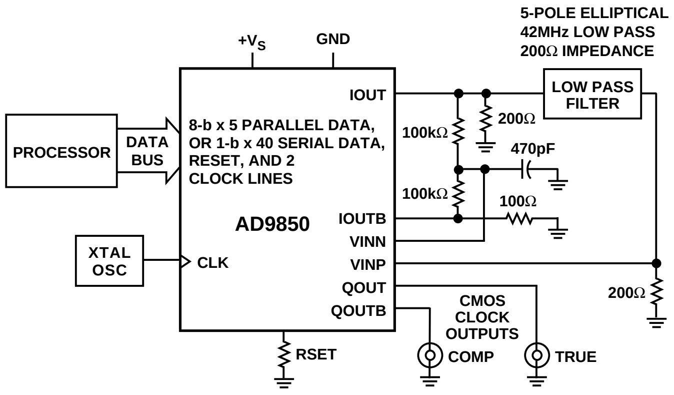
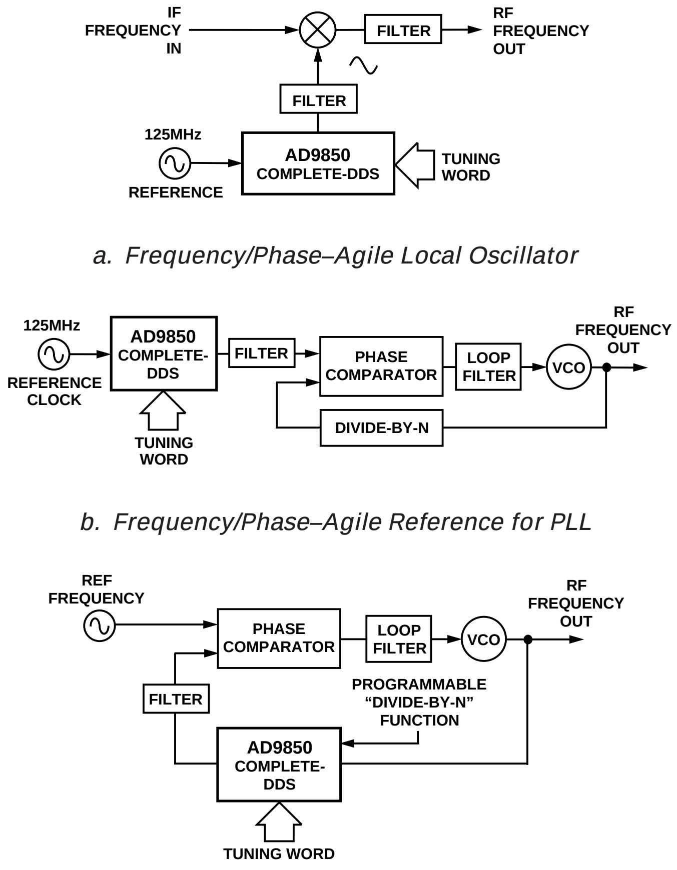


Figure 13. Basic AD9850 Clock Generator Application with Low-Pass Filter



a. Frequency/Phase–Agile Local Oscillator

b. Frequency/Phase–Agile Reference for PLL

c. Digitally-Programmable ”Divide-by-N“ Function in PLL

Figure 15. AD9850 Complete-DDS Synthesizer in Frequency Up-Conversion Applications

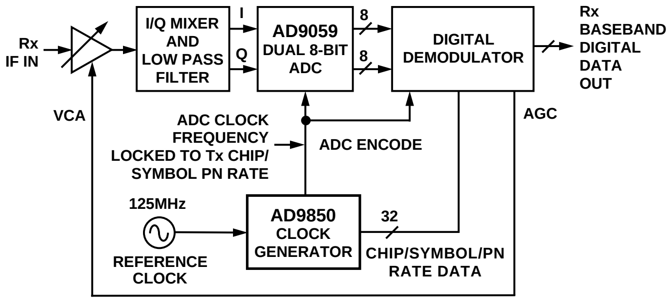


Figure 14. AD9850 Clock Generator Application in a Spread-Spectrum Receiver

THEORY OF OPERATION AND APPLICATION

The AD9850 uses direct digital synthesis (DDS) technology, in the form of a numerically controlled oscillator, to generate a frequency/phase-agile sine wave. The digital sine wave is converted to analog form via an internal 10-bit high speed D/A converter, and an onboard high speed comparator is provided to translate the analog sine wave into a low-jitter TTL/CMOS-compatible output square wave. DDS technology is an innovative circuit architecture that allows fast and precise manipulation of its output frequency under full digital control. DDS also enables very high resolution in the incremental selection of output frequency; the AD9850 allows an output frequency resolution of 0.0291 Hz with a 125 MHz reference clock applied. The AD9850's output waveform is phase-continuous when changed.

The basic functional block diagram and signal flow of the AD9850 configured as a clock generator is shown in Figure 16.

The DDS circuitry is basically a digital frequency divider function whose incremental resolution is determined by the frequency of the reference clock divided by the 2^N number of bits in the tuning word. The phase accumulator is a variable-modulus counter that increments the number stored in it each time it receives a clock pulse. When the counter overflows it wraps around, making the phase accumulator's output contiguous. The frequency tuning word sets the modulus of the counter that effectively determines the size of the increment (Δ Phase) that gets added to the value in the phase accumulator on the next clock pulse. The larger the added increment, the faster the accumulator overflows, which results in a higher output frequency. The AD9850 uses an innovative and proprietary algorithm that mathematically converts the 14-bit truncated value of the phase accumulator to the appropriate COS value. This unique algorithm uses a much reduced ROM look-up table and DSP techniques to perform this function, which contributes to the small size and low power dissipation of the AD9850. The relationship of the output frequency, reference clock, and tuning word of the AD9850 is determined by the formula:

$$F_{OUT} = (\Delta\ Phase \times CLKIN)/2^{32}$$

where: $\Delta\ Phase$ = value of 32-bit tuning word
$CLKIN$ = input reference clock frequency in MHz
F_{OUT} = frequency of the output signal in MHz

The digital sine wave output of the DDS block drives the internal high speed 10-bit D/A converter that reconstructs the sine wave in analog form. This DAC has been optimized for dynamic

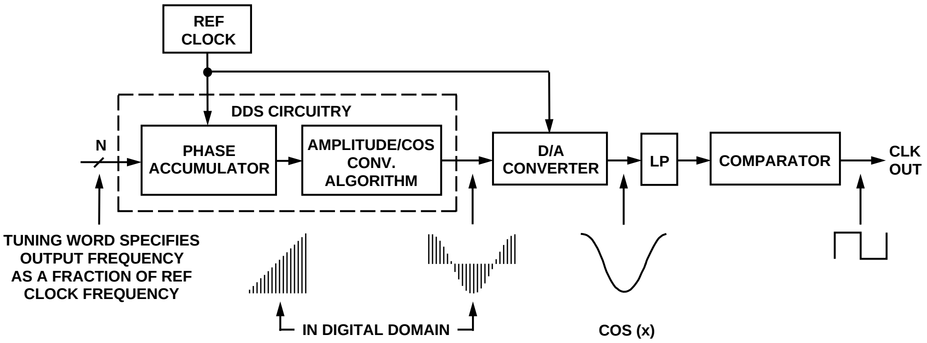

REF CLOCK
DDS CIRCUITRY
N
PHASE ACCUMULATOR
AMPLITUDE/COS CONV. ALGORITHM
D/A CONVERTER
LP
COMPARATOR
CLK OUT
TUNING WORD SPECIFIES OUTPUT FREQUENCY AS A FRACTION OF REF CLOCK FREQUENCY
IN DIGITAL DOMAIN
COS (x)

Figure 16. Basic DDS Block Diagram and Signal Flow of AD9850

performance and low glitch energy as manifested in the low jitter performance of the AD9850. Since the output of the AD9850 is a sampled signal, its output spectrum follows the Nyquist sampling theorem. Specifically, its output spectrum contains the fundamental plus aliased signals (images) that occur at multiples of the Reference Clock Frequency ± the selected output frequency. A graphical representation of the sampled spectrum, with aliased images, is shown in Figure 17.

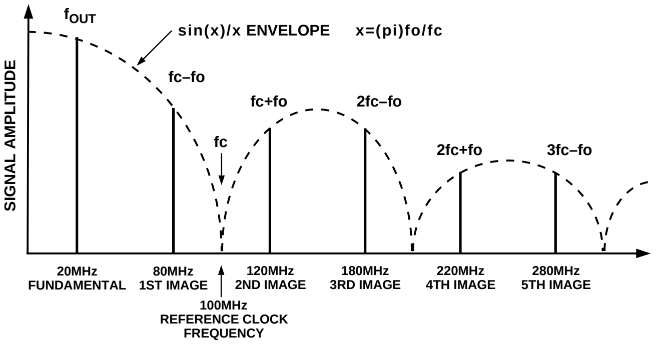


Figure 17. Output Spectrum of a Sampled Signal

In this example, the reference clock is 100 MHz and the output frequency is set to 20 MHz. As can be seen, the aliased images are very prominent and of a relatively high energy level as determined by the sin(x)/x roll-off of the quantized D/A converter output. In fact, depending on the fo/Ref Clk relationship, the first aliased image can be on the order of –3 dB below the fundamental. A low-pass filter is generally placed between the output of the D/A converter and the input of the comparator to further suppress the effects of aliased images. Obviously, consideration must be given to the relationship of the selected output frequency and the Reference Clock frequency to avoid unwanted (and unexpected) output anomalies.

A good rule-of-thumb for applying the AD9850 as a clock generator is to limit the selected output frequency to <33% of Reference Clock frequency, thereby avoiding generating aliased signals that fall within, or close to, the output band of interest (generally dc-selected output frequency). This practice will ease the complexity (and cost) of the external filter requirement for the clock generator application.

The reference clock frequency of the AD9850 has a minimum limitation of 1 MHz. The device has internal circuitry that senses when the minimum clock rate threshold has been exceeded and automatically places itself in the power-down mode. When in this state, if the clock frequency again exceeds the threshold, the device resumes normal operation. This shutdown mode prevents excessive current leakage in the dynamic registers of the device.

The D/A converter output and comparator inputs are available as differential signals that can be flexibly configured in any manner desired to achieve the objectives of the end-system. The typical application of the AD9850 is with single-ended output/input analog signals, a single low-pass filter, and generating the comparator reference midpoint from the differential DAC output as shown in Figure 13.

Programming the AD9850

The AD9850 contains a 40-bit register that is used to program the 32-bit frequency control word, the 5-bit phase modulation word and the power-down function. This register can be loaded in a parallel or serial mode.

In the parallel load mode, the register is loaded via an 8-bit bus; the full 40-bit word requires five iterations of the 8-bit word. The W_CLK and FQ_UD signals are used to address and load the registers. The rising edge of FQ_UD loads the (up to) 40-bit control data word into the device and resets the address pointer to the first register. Subsequent W_CLK rising edges load the 8-bit data on words [7:0] and move the pointer to the next register. After *five* loads, W_CLK edges are ignored until either a reset or an FQ_UD rising edge resets the address pointer to the first register.

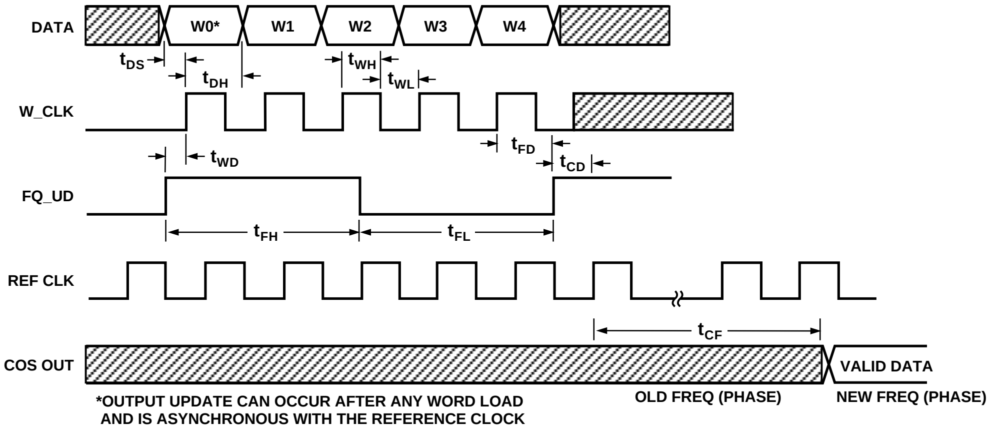
In serial load mode, subsequent rising edges of W_CLK shift the 1-bit data on Lead 25 (D7) through the 40 bits of programming information. After 40 bits are shifted through, an FQ_UD pulse is required to update the output frequency (or phase).

The function assignments of the data and control words are shown in Table III; the detailed timing sequence for updating the output frequency and/or phase, resetting the device, and powering-up/down, are shown in the timing diagrams of Figures 18–24.

Note: There are specific control codes, used for factory test purposes, that render the AD9850 temporarily inoperable. The user must take deliberate precaution to avoid inputting the codes listed in Table II.

Table II. Factory-Reserved Internal Test Control Codes

Loading Format	Factory-Reserved Codes
Parallel	1) W0 = XXXXXX10 2) W0 = XXXXXX01
Serial	1) W32 = 1; W33 = 0 2) W32 = 0; W33 = 1 3) W32 = 1, W33 = 1

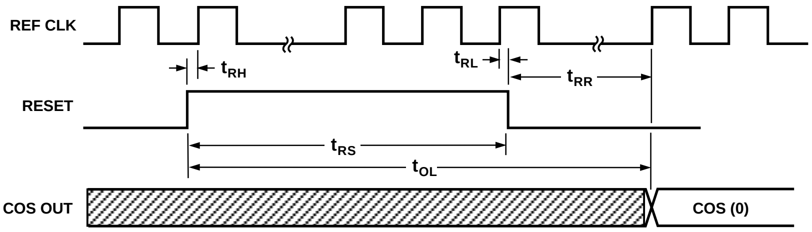


SYMBOL	DEFINITION	MIN
t_{DS}	DATA SETUP TIME	3.5ns
t_{DH}	DATA HOLD TIME	3.5ns
t_{WH}	W_CLK HIGH	3.5ns
t_{WL}	W_CLK LOW	3.5ns
t_{WD}	W_CLK DELAY AFTER FQ_UD	7.0ns
t_{CD}	CLK DELAY AFTER FQ_UD	3.5ns
t_{FH}	FQ_UD HIGH	7.0ns
t_{FL}	FQ_UD LOW	7.0ns
t_{FD}	FQ_UD DELAY AFTER W_CLK	7.0ns
t_{CF}	OUTPUT LATENCY FROM FQ_UD	
	FREQUENCY CHANGE	18 CLOCK CYCLES
	PHASE CHANGE	13 CLOCK CYCLES

Figure 18. Parallel-Load Frequency/Phase Update Timing Sequence

Table III. 8-Bit Parallel-Load Data/Control Word Functional Assignment

Word	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
W0	Phase-b4 (MSB)	Phase-b3	Phase-b2	Phase-b1	Phase-b0 (LSB)	Power-Down	Control	Control
W1	Freq-b31 (MSB)	Freq-b30	Freq-b29	Freq-b28	Freq-b27	Freq-b26	Freq-b25	Freq-b24
W2	Freq-b23	Freq-b22	Freq-b21	Freq-b20	Freq-b19	Freq-b18	Freq-b17	Freq-b16
W3	Freq-b15	Freq-b14	Freq-b13	Freq-b12	Freq-b11	Freq-b10	Freq-b9	Freq-b8
W4	Freq-b7	Freq-b6	Freq-b5	Freq-b4	Freq-b3	Freq-b2	Freq-b1	Freq-b0 (LSB)



SYMBOL	DEFINITION	MIN SPEC
t_{RH}	CLK DELAY AFTER RESET RISING EDGE	3.5ns
t_{RL}	RESET FALLING EDGE AFTER CLK	3.5ns
t_{RR}	RECOVERY FROM RESET	2 CLK CYCLES
t_{RS}	MINIMUM RESET WIDTH	5 CLK CYCLES
t_{OL}	RESET OUTPUT LATENCY	13 CLK CYCLES

RESULTS OF RESET:
- FREQUENCY/PHASE REGISTER SET TO 0
- ADDRESS POINTER RESET TO W0
- POWER-DOWN BIT RESET TO “0”
- DATA INPUT REGISTER UNEFFECTED

Figure 19. Master Reset Timing Sequence

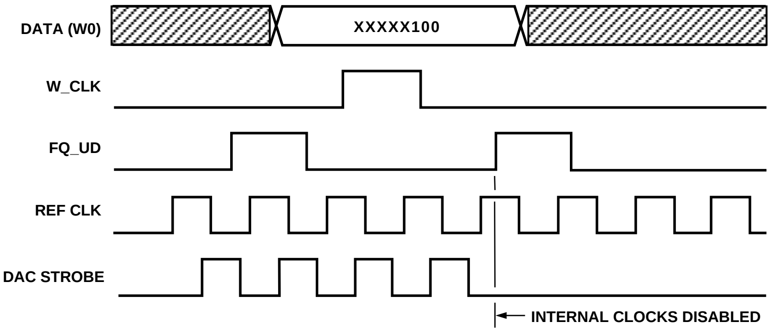


Figure 20. Parallel-Load Power-Down Sequence/Internal Operation

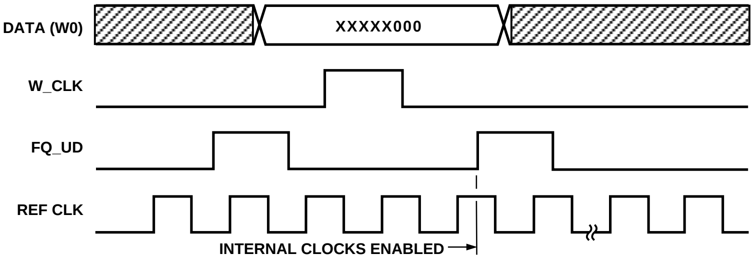


Figure 21. Parallel-Load Power-Up Sequence/Internal Operation

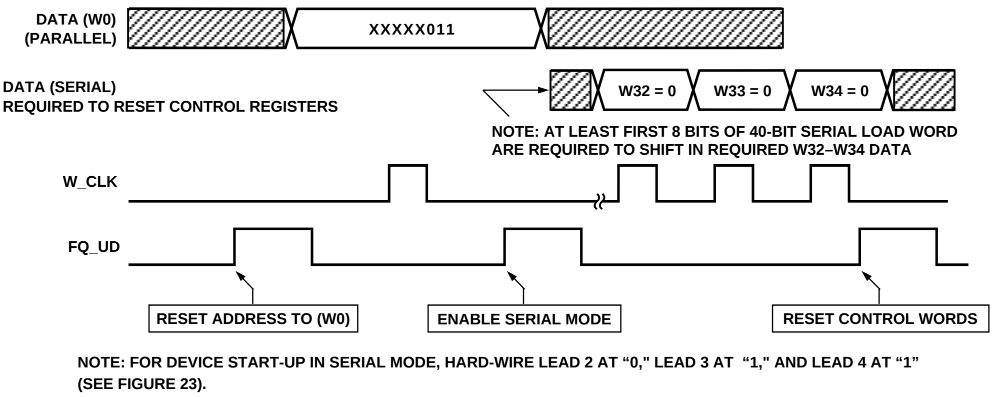

DATA (W0)
(PARALLEL)
XXXXX011
DATA (SERIAL)
REQUIRED TO RESET CONTROL REGISTERS
W32 = 0
W33 = 0
W34 = 0
NOTE: AT LEAST FIRST 8 BITS OF 40-BIT SERIAL LOAD WORD ARE REQUIRED TO SHIFT IN REQUIRED W32–W34 DATA
W_CLK
FQ_UD
RESET ADDRESS TO (W0)
ENABLE SERIAL MODE
RESET CONTROL WORDS
NOTE: FOR DEVICE START-UP IN SERIAL MODE, HARD-WIRE LEAD 2 AT “0,” LEAD 3 AT “1,” AND LEAD 4 AT “1” (SEE FIGURE 23).

Figure 22. Serial-Load Enable Sequence

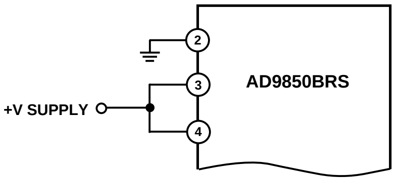


Figure 23. Leads 2–4 Connection for Default Serial-Mode Operation

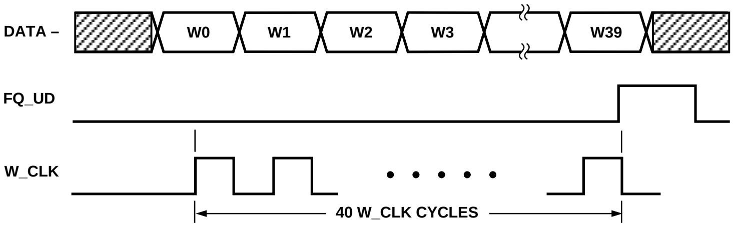


Figure 24. Serial-Load Frequency/Phase Update Sequence

Table IV. 40-Bit Serial-Load Word Function Assignment

W0	Freq-b0 (LSB)	W14	Freq-b14	W28	Freq-b28
W1	Freq-b1	W15	Freq-b15	W29	Freq-b29
W2	Freq-b2	W16	Freq-b16	W30	Freq-b30
W3	Freq-b3	W17	Freq-b17	W31	Freq-b31 (MSB)
W4	Freq-b4	W18	Freq-b18	W32	Control
W5	Freq-b5	W19	Freq-b19	W33	Control
W6	Freq-b6	W20	Freq-b20	W34	Power-Down
W7	Freq-b7	W21	Freq-b21	W35	Phase-b0 (LSB)
W8	Freq-b8	W22	Freq-b22	W36	Phase-b1
W9	Freq-b9	W23	Freq-b23	W37	Phase-b2
W10	Freq-b10	W24	Freq-b24	W38	Phase-b3
W11	Freq-b11	W25	Freq-b25	W39	Phase-b4 (MSB)
W12	Freq-b12	W26	Freq-b26		
W13	Freq-b13	W27	Freq-b27		

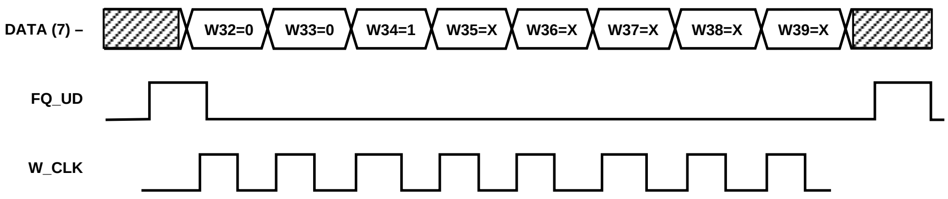
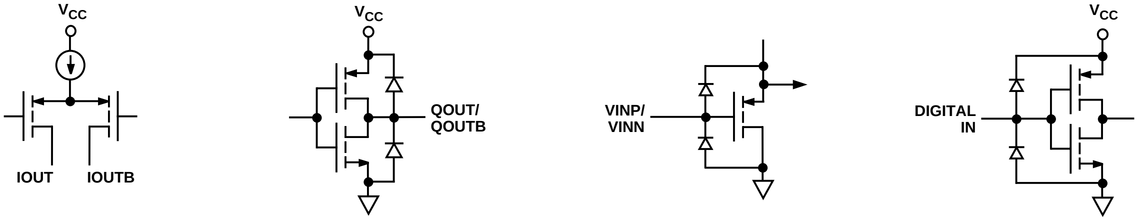


Figure 25. Serial-Load Power-Down Sequence



DAC Output *Comparator Output* *Comparator Input* *Digital Inputs*

Figure 26. AD9850 I/O Equivalent Circuits

PCB LAYOUT INFORMATION

The AD9850/CGPCB and AD9850/FSPCB evaluation boards (Figures 27–30) represent typical implementations of the AD9850 and exemplify the use of high frequency/high resolution design and layout practices. The printed circuit board that contains the AD9850 should be a multilayer board that allows dedicated power and ground planes. The power and ground planes should be free of etched traces that cause discontinuities in the planes. It is recommended that the top layer of the multilayer board also contain interspatial ground plane, which makes ground available for surface-mount devices. If separate analog and digital system ground planes exist, they should be connected together at the AD9850 for optimum results.

Avoid running digital lines under the device as these will couple noise onto the die. The power supply lines to the AD9850 should use as large a track as possible to provide a low-impedance path and reduce the effects of glitches on the power supply line. Fast switching signals like clocks should be shielded with ground to avoid radiating noise to other sections of the board. Avoid crossover of digital and analog signal paths. Traces on opposite sides of the board should run at right angles to each other. This will reduce the effects of feedthrough through the circuit board. Use microstrip techniques where possible.

Good decoupling is also an important consideration. The analog (AVDD) and digital (DVDD) supplies to the AD9850 are independent and separately pinned out to minimize coupling between analog and digital sections of the device. All analog and digital supplies should be decoupled to AGND and DGND, respectively, with high quality ceramic capacitors. To achieve best performance from the decoupling capacitors, they should be placed as close as possible to the device, ideally right up against the device. In systems where a common supply is used to drive both the AVDD and DVDD supplies of the AD9850, it is recommended that the system's AVDD supply be used.

Analog Devices, Inc., applications engineering support is available to answer additional questions on grounding and PCB layout. Call 1-800-ANALOGD.

Evaluation Boards

Two versions of evaluation boards are available for the AD9850, which facilitate the implementation of the device for benchtop analysis, and serve as a reference for PCB layout. The *AD9850/FSPCB* is intended for applications where the device will primarily be used as frequency synthesizer. This version facilitates connection of the AD9850's internal D/A converter output to a 50 Ω spectrum analyzer input; the internal comparator on the AD9850 DUT is not enabled (see Figure 28 for electrical schematic of AD9850/FSPCB). The *AD9850/CGPCB* is intended for applications using the device in the clock generator mode. It connects the AD9850's DAC output to the internal comparator input via a single-ended, 42 MHz low-pass, 5-pole Elliptical filter. This model facilitates the access of the AD9850's comparator output for evaluation of the device as a frequency- and phase-agile clock source (see Figure 29 for electrical schematic of AD9850/CGPCB).

Both versions of the AD9850 evaluation boards are designed to interface to the parallel printer port of a PC. The operating software runs under Microsoft® Windows and provides a user-friendly and intuitive format for controlling the functionality and observing the performance of the device. The 3.5" floppy provided with the evaluation board contains an executable file that loads and displays the AD9850 function-selection screen. The evaluation board may be operated with +3.3 V or +5 V supplies. The evaluation boards are configured at the factory for an external reference clock input; if the onboard crystal clock source is used, remove R2.

All trademarks are the property of their respective holders.

AD9850 Evaluation Board Instructions

Required hardware/software:

IBM compatible computer operating in a Windows environment

Printer port, 3.5" floppy drive and Centronics compatible printer cable.

XTAL clock or signal generator—if using a signal generator, dc offset the signal to one-half the supply voltage and apply at least 3 V p-p signal across the 50 Ω (R2) input resistor. Remove R2 for high Z clock input.

AD9850 evaluation board software disk and AD9850/FSPCB or AD9850/CGPCB evaluation board.

+5 V voltage supply

Setup:

Copy the contents of the AD9850 disk onto your hard drive (there are three files).

Connect the printer cable from computer to the AD9850 evaluation board.

Apply power to AD9850 evaluation board. The AD9850 is powered separately from the connector marked "DUT +V."

The AD9850 may be powered with 3.3 V to +5 V.

Connect external 50 ohm clock or remove R2 and apply a high Z input clock such as a crystal "can" oscillator.

Locate the file called 9850REV2.EXE and execute that program.

Monitor should display a "control panel" to allow operation of the AD9850 evaluation board.

Operation:

On the control panel, locate the box called "COMPUTER I/O." Point to and click the selection marked LPT1 and then point to the "TEST" box and click. A message will appear telling you if your choice of output ports is correct. Choose other ports as necessary to achieve a correct setting. If you have trouble getting your computer to recognize any printer port, try the following: connect three 2K pull-up resistors from Pins 9, 8 and 7 of U3 to +5 V. This will assist "weak" printer port outputs in driving the heavy capacitance load of the printer cable. If troubles persist, try a different printer cable.

Locate the "MASTER RESET" button with the mouse and click it. This will reset the AD9850 to 0 Hz, 0 degrees phase. The output should be a dc voltage equal to the full-scale output of the AD9850.

Locate the "CLOCK" box and place the cursor in the frequency box. Type in the clock frequency (in MHz) that you will be applying to the AD9850. Click the LOAD button or press enter on the keyboard.

Move the cursor to the OUTPUT FREQUENCY box and type in the desired output frequency (in MHz). Click the "LOAD" button or press the enter key. The BUS MONITOR section of the control panel will show the 32-bit word that was loaded into the AD9850. Upon completion of this step, the AD9850 output should be active and outputting your frequency information.

Changing the output phase is accomplished by clicking on the "down arrow" in the OUTPUT PHASE DELAY box to make a selection and then clicking the LOAD button.

Other operational modes (Frequency Sweeping, Sleep, Serial Input) are available to the user via keyboard/mouse control.

The AD9850/FSPCB provides access into and out of the on-chip comparator via test point pairs (each pair has an active input and a ground connection). The two active inputs are labeled TP1 and TP2. The unmarked hole next to each labeled test point is a ground connection. The two active outputs are labeled TP5 and TP6. Unmarked ground connections are adjacent to each of these test points .

The AD9850/CGPCB provides BNC inputs and outputs associated with the on-chip comparator and the onboard, 5th order, 200 ohm input/output Z, elliptic 45 MHz low-pass filter. Jumpering (soldering a wire) E1 to E2, E3 to E4, and E5 to E6 connects the onboard filter and the midpoint switching voltage to the comparator. Users may elect to insert their own filter and comparator threshold voltage by removing the jumpers and inserting a filter between J7 and J6 and then providing a threshold voltage at E1.

If you choose to use the XTAL socket to supply the clock to the AD9850, you must remove R2 (a 50 ohm chip resistor). The crystal oscillator must be either TTL or CMOS (preferably) compatible.

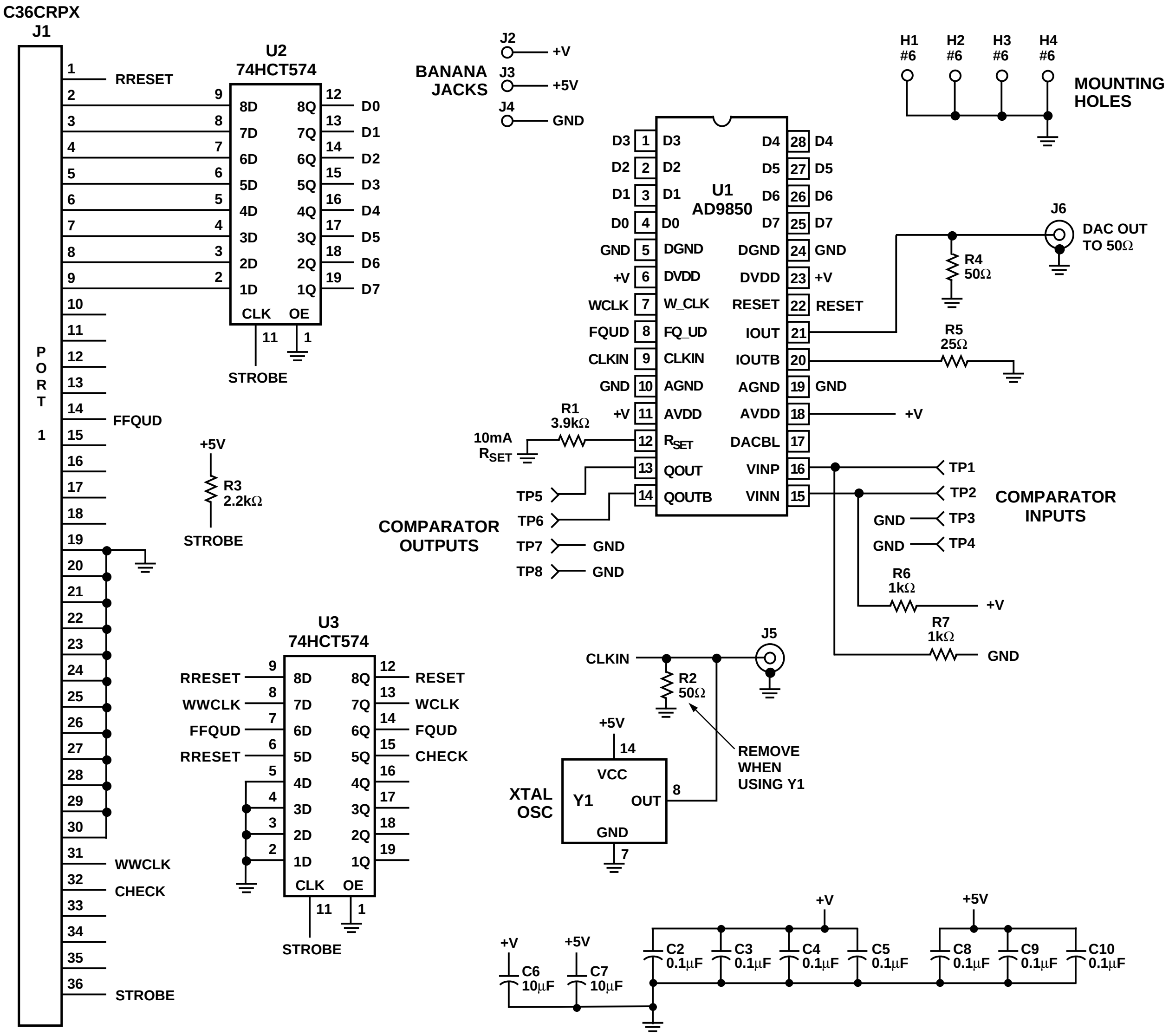


Figure 27. AD9850/FSPCB Electrical Schematic

COMPONENT LIST

Integrated Circuits

U1	AD9850BRS (28-Pin SSOP)
U2, U3	74HCT574 H-CMOS Octal Flip-Flop

Capacitors

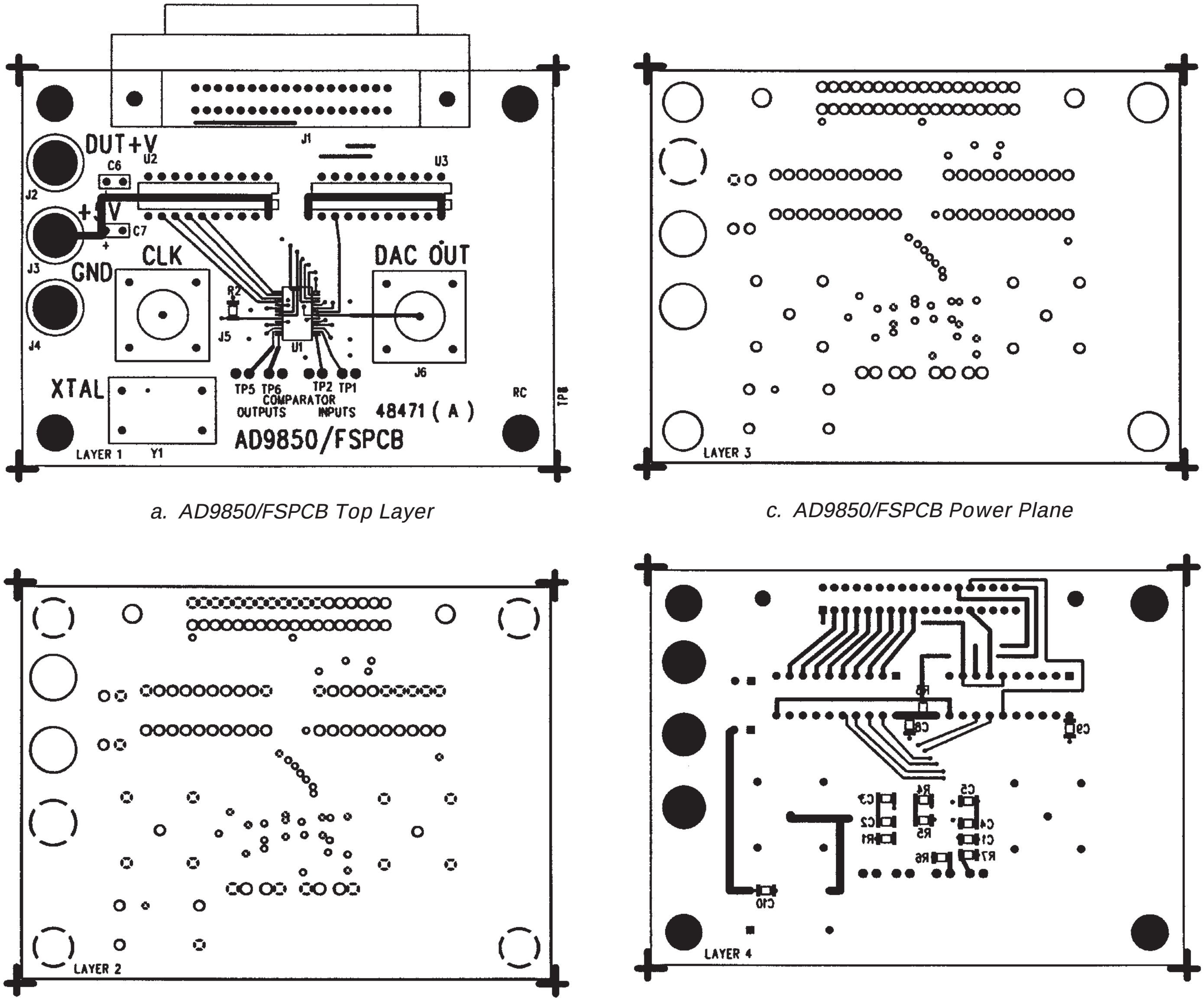
C2–C5, C8–C10	0.1 µF Ceramic Chip Capacitor
C6, C7	10 µF Tantalum Chip Capacitor

Resistors

R1	3.9 kΩ Resistor
R2, R4	50 Ω Resistor
R3	2.2 kΩ Resistor
R5	25 Ω Resistor
R6, R7	1 kΩ Resistor

Connectors

J1	36-Pin D Connector
J2, J3, J4	Banana Jack
J5, J6	BNC Connector



a. AD9850/FSPCB Top Layer

c. AD9850/FSPCB Power Plane

b. AD9850/FSPCB Ground Plane

d. AD9850/FSPCB Bottom Layer

Figure 28. AD9850/FSPCB Evaluation Board Layout

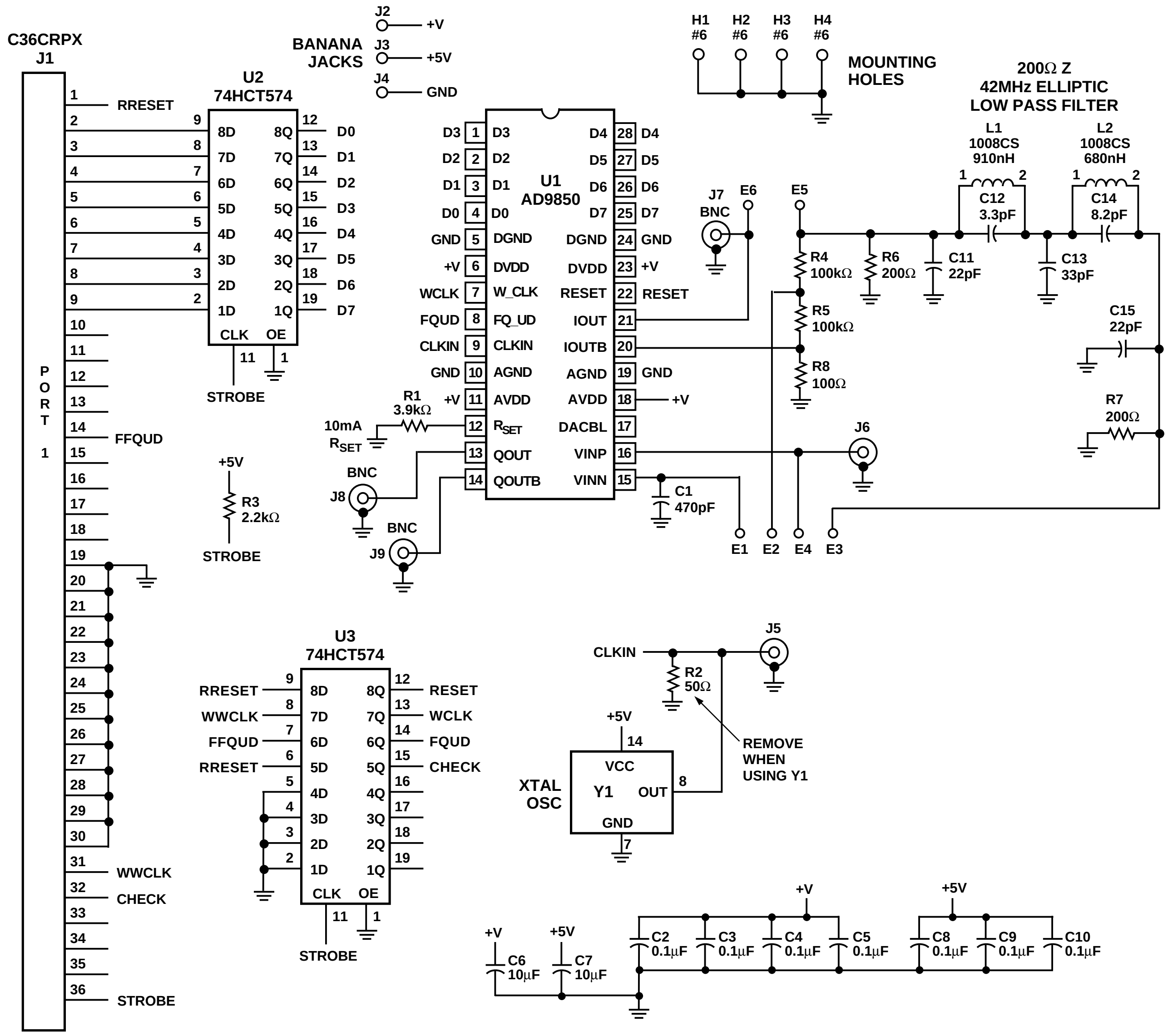


Figure 29. AD9850/CGPCB Electrical Schematic

COMPONENT LIST

Integrated Circuits

U1	AD9850BRS (28-Pin SSOP)
U2, U3	74HCT574 H-CMOS Octal Flip-Flop

Capacitors

C1	470 pF Ceramic Chip Capacitor
C2–C5, C8–C10	0.1 µF Ceramic Chip Capacitor
C6, C7	10 µF Tantalum Chip Capacitor
C11	22 pF Ceramic Chip Capacitor
C12	3.3 pF Ceramic Chip Capacitor
C13	33 pF Ceramic Chip Capacitor
C14	8.2 pF Ceramic Chip Capacitor
C15	22 pF Ceramic Chip Capacitor

Resistors

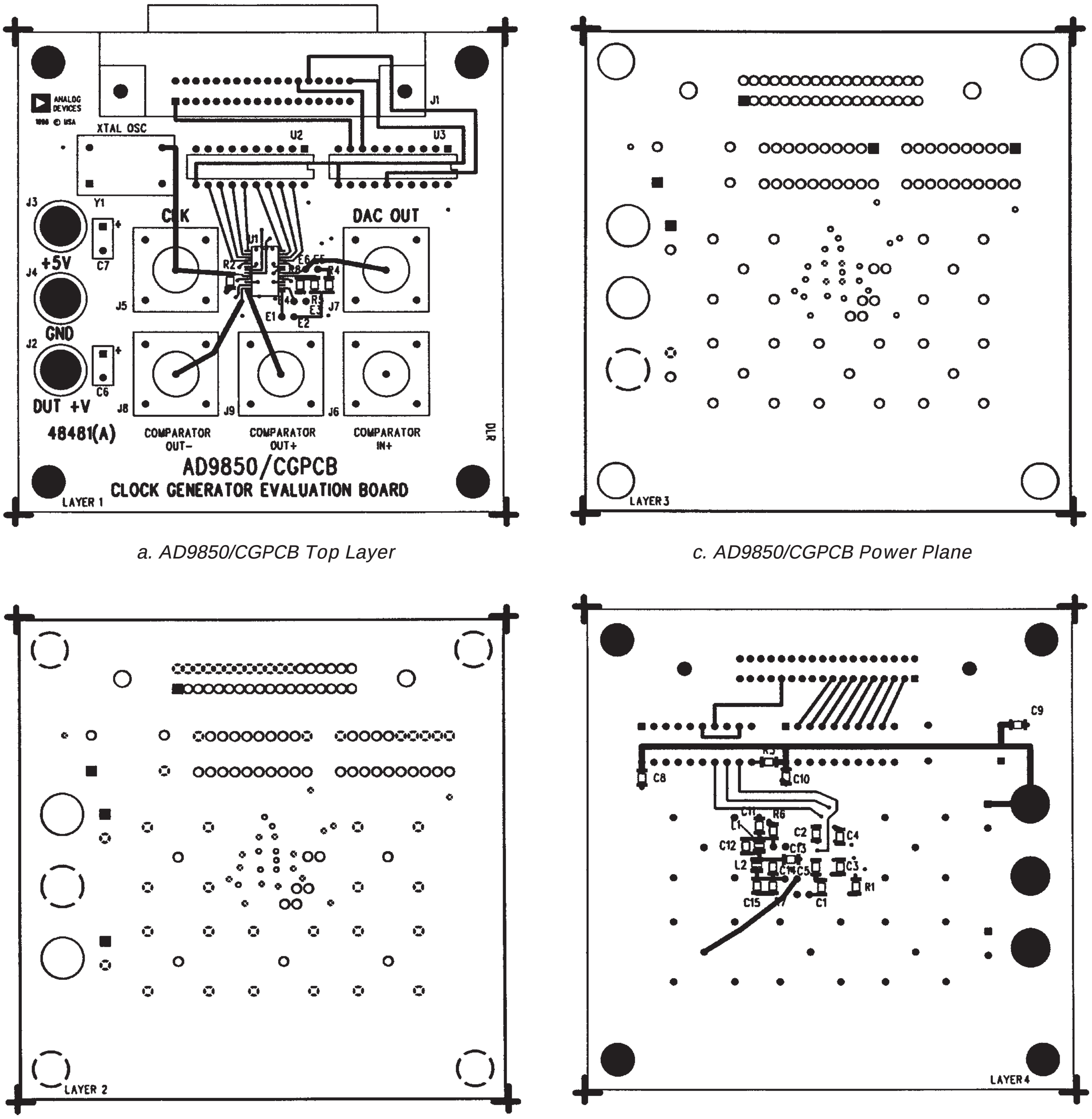
R1	3.9 kΩ Resistor
R2	50 Ω Resistor
R3	2.2 kΩ Resistor
R4, R5	100 kΩ Resistor
R6, R7	200 Ω Resistor
R8	100 Ω Resistor

Connectors

J2, J3, J4	Banana Jack
J5–J9	BNC Connector

Inductors

L1	910 nH Surface Mount
L2	680 nH Surface Mount



a. AD9850/CGPCB Top Layer

c. AD9850/CGPCB Power Plane

b. AD9850/CGPCB Ground Plane

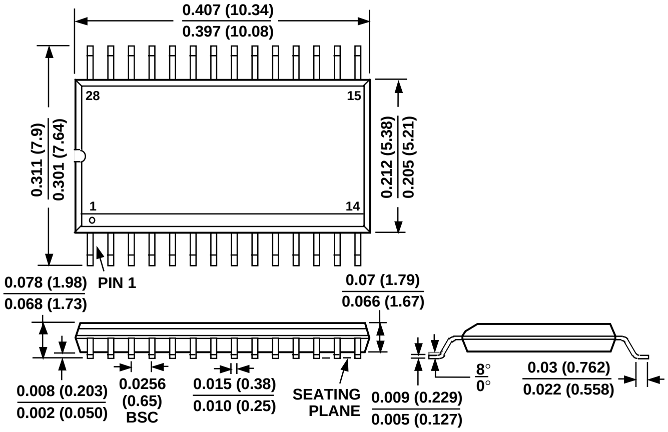
d. AD9850/CGPCB Bottom Layer

Figure 30. AD9850/CGPCB Evaluation Board Layout

OUTLINE DIMENSIONS

Dimensions shown in inches and (mm).

28-Lead Shrink Small Outline Package (RS-28)



PRINTED IN U.S.A.

V360EPC Rev. A0

LOCAL BUS TO PCI BRIDGE
FOR DE-MULTIPLEXED A/D PROCESSORS

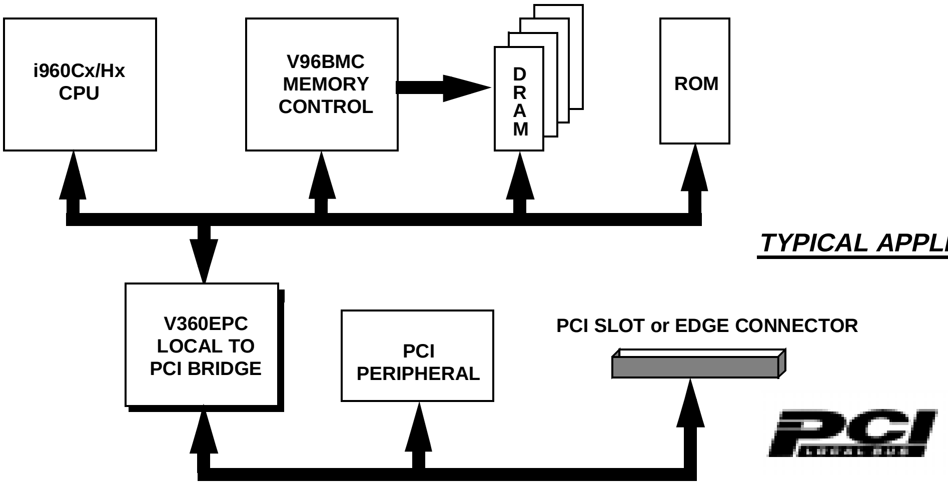
- Glueless i960Cx/Hx and AMD29030/40 processors interface
- Large, 640-byte FIFOs using V3's unique *DYNAMIC BANDWIDTH ALLOCATION™* architecture
- Increased 64-byte Read FIFO per aperture
- Supports both target and master modes
- Dual bi-directional address space remapping
- Fully compliant with PCI 2.1 specification
- On-the-fly byte order (endian) conversion
- I_2O ready hardware messaging unit
- 2 channel DMA controller plus Multiprocessor DMA chaining and demand mode DMA
- Hot Swapping Capability
- Bi-directional mailboxes w/doorbell interrupts
- Flexible PCI and local interrupt management
- Serial EEPROM configuration interface
- 33MHz and 50MHz local bus versions
- Low cost 160-pin EIAJ PQFP package

V360EPC provides the highest performance, most flexible, and most economical method to directly connect i960Cx/Hx or AMD2930/40 processors to the PCI bus. As a generic solution for 32-bit de-multiplexed local bus applications, V360EPC is also a suitable candidate for a variety of high-performance applications based on Motorola, IBM, DEC and Hitachi embedded processors - where a minimal amount of glue logic is only needed.

V360EPC is a second generation of V3's I_2O ready PCI bridges - fully backward compatible with V962PBC and V292PBC Rev B2 devices - and is supporting powerful features like Hot Swapping and DMA chaining. The PCI bus can be run at full 33MHz, regardless of processor clock rate. The overall throughput of the system is dramatically improved by increasing the FIFO depths and utilizing the unique *DYNAMIC BANDWIDTH ALLOCATION™* architecture.

Access to the PCI bus can be performed through two programmable address apertures. Two more apertures are provided for PCI-to-local bus accesses. There are 64-bytes of read FIFOs dedicated for each aperture in each direction.

Two high-performance DMA channels with chaining and demand mode capabilities provide a powerful data transfer engine for bulk data transfers. Mailbox registers and flexible PCI interrupt controlers are also included to provide a simple mechanism to emulate PCI device control ports. The part is available in 160-pin low cost PQFP packages.



TYPICAL APPLICATION

V3 Semiconductor reserves the right to change the specifications of this product without notice.
V360EPC and V96BMC are trademarks of V3 Semiconductor. All other trademarks are the property of their respective owners.

This document contains the product codes, pinouts, package mechanical information, DC characteristics, and AC characteristics for the V360EPC. Detailed functional information is contained in the User's Manual.

V3 Semiconductor retains the rights to change documentation, specifications, or device functionality at any time without notice. Please verify that you have the latest copy of all documents before finalizing a design.

1.0 Product Codes

Table 1: Product Codes

Product Code	Processors	Bus Type	Package	Frequency
V360EPC-33 REVA0	i960Cx/Hx, AMD29030/40	32-bit demultiplexed	160-pin EIAJ PQFP	33MHz
V360EPC-50 REV A0	i960Cx/Hx, AMD29030/40	32-bit demultiplexed	160-pin EIAJ PQFP	50MHz

2.0 Pin Description and Pinout

Table 2 below lists the pin types found on the V360EPC. Table 3 describes the function of each pin on the V360EPC. Table 5 lists the pins by pin number. Figure 1 shows the pinout for the 160-pin EIAJ PQFP package and Figure 2 shows the mechanical dimensions of the package.

Table 2: Pin Types

Pin Type	Description
PCI I	PCI input only pin.
PCI O	PCI output only pin.
PCI I/O	PCI tri-state I/O pin.
PCI I/OD	PCI input with open drain output.
I/O_4	TTL I/O pin with 4mA output drive.
I	TTL input only pin.
O_4	TTL output pin with 4mA output drive.

Table 3: Signal Descriptions

PCI Bus Interface			
Signal	Type	R[a]	Description
AD[31:0]	PCI I/O	Z	Address and data, multiplexed on the same pins.
C/$\overline{\text{BE[3:0]}}$	PCI I/O	Z	Bus Command and Byte Enables, multiplexed on the same pins.
PAR	PCI I/O	Z	Parity represents even parity across AD[31:0] and C/$\overline{\text{BE[3:0]}}$.
$\overline{\text{FRAME}}$	PCI I/O	Z	Cycle Frame indicates the beginning and burst length of an access.
$\overline{\text{IRDY}}$	PCI I/O	Z	Initiator Ready indicates the initiating agent's (bus master's) ability to complete the current data phase of the transaction.
$\overline{\text{TRDY}}$	PCI I/O	Z	Target Ready indicates the target agent's (selected device's) ability to complete the current data phase of the transaction.
$\overline{\text{STOP}}$	PCI I/O	Z	Stop indicates the current target is requesting the master to stop the current transaction (retry or disconnect).
$\overline{\text{DEVSEL}}$	PCI I/O	Z	Device Select, when actively driven by a target, indicates the driving device has decoded its address as the target of the current access. As an input to the initiator, $\overline{\text{DEVSEL}}$ indicates whether any device on the bus has been selected.
IDSEL	PCI I		Initialization Device Select is used as a chip select during configuration read and write transactions. It must be driven high in order to access the chip's internal configuration space.
$\overline{\text{REQ}}$	PCI O	Z	Request indicates to the arbiter that this agent requests use of the bus.
$\overline{\text{GNT}}$	PCI I		Grant indicates to the agent that access to the bus has been granted.
PCLK	PCI I		PCLK provides timing for all transactions on the PCI bus.
$\overline{\text{PRST}}$	PCI I/O	Z/L	Acts as an input when RDIR is high, an output when RDIR is low. As an input it is asserted low to bring all internal PBC operation to a reset state.
$\overline{\text{PERR}}$	PCI I/O	Z	Parity Error is used to report data parity errors during all PCI transactions except a Special Cycle.
$\overline{\text{SERR}}$	PCI I/OD	Z	System Error is used to report address parity errors, data parity errors on the Special Cycle command, or any other system error where the result will be catastrophic.
$\overline{\text{INT[A:D]}}$	PCI I/OD	Z	Level-sensitive interrupt requests may be received or generated.

Table 3: Signal Descriptions (cont'd)

Local Bus Interface			
Signal	Type	R	Description
LD[31:0] ID[31:0][b]	I/O4	Z	Local multiplexed address and data bus.
LA[31:2]	I/O4	Z	Local address bus.
$\overline{\text{BE[3:0]}}$ $\overline{\text{BWE[3:0]}}$[b]	I/O4	Z	Local bus byte enables.
W/$\overline{\text{R}}$ R/$\overline{\text{W}}$[b]	I/O4	Z	Read-Write strobe.
$\overline{\text{ADS}}$ $\overline{\text{LREQ}}$[b]	I/O4	Z	Asserted low to indicate the beginning of a bus cycle.
$\overline{\text{READY}}$ $\overline{\text{RDY}}$[c]	I/O4	Z	Local Bus data ready
HOLD $\overline{\text{LBREQ}}$[b]	O4	L	Local bus hold request: asserted by the chip to initiate a local bus master cycle.
HOLDA $\overline{\text{LBGRT}}$[b]	I		Local bus hold acknowledge.
LPAR[3:0]	I/O4	Z	Local bus parity.
$\overline{\text{BLAST}}$ $\overline{\text{BURST}}$[b]	I/O4	Z	Burst last[c]. Burst request[b].
$\overline{\text{BTERM}}$ $\overline{\text{ERR}}$[b]	I/O4	Z	Bus Time-out. Burst terminate[c].
$\overline{\text{LINT}}$	O4	H	Local interrupt request.
$\overline{\text{LRST}}$	I/O4	L/Z	Local bus RESET signal.
LCLK MEMCLK[b]	I		Local bus clock.

Serial EEPROM Interface			
Signal	Type	R	Description
SCL/$\overline{\text{LPERR}}$	O4	X	EEPROM clock. Local parity error.
SDA	I/O4	X	EEPROM data.

Table 3: Signal Descriptions (cont'd)

Configuration			
Signal	Type	R	Description
RDIR	I		Reset direction. Tie low to drive $\overline{\text{PRST}}$ out and $\overline{\text{LRST}}$ in, high to drive $\overline{\text{LRST}}$ out and $\overline{\text{PRST}}$ in.
$\overline{\text{EN5V}}$	I		Selects 5V ($\overline{\text{EN5V}}$ driven low) or 3.3V ($\overline{\text{EN5V}}$ driven high) device operation modes.
Power and Ground Signals			
Signal	Type	R	Description
V_{CC}	-		POWER leads intended for external connection to a V_{CC} board plane.
GND	-		GROUND leads intended for external connection to a GND board plane.

a. **R** indicates state during reset.
b. Applies to AMD29030/40 mode.
c. Applies to i960Cx/Hx mode.

2.1 Test Mode Pins

Several device pins are used during manufacturing test to put the V360EPC device into various test modes. ***These pins must be maintained at proper levels during reset to insure proper operation.*** This is typically handled through pull-up or pull-down resistors (typically 1K to 10K) on the signal pins if they are not guaranteed to be at the proper level during reset. Table 4 below shows the reset states for test mode pins:

Table 4: RESET State for Test Mode Pins

Mode	Pin 134	Pin 135	Pin 153
i960Cx/Hx	Pull-Up	Pull-Up	Pull-Up
AMD2930/40	Pull-Down	Pull-Up	Pull-Up

Table 5: Pin Assignments

PIN #	Signal	PIN #	Signal	PIN #	Signal	PIN #	Signal
1	V_{CC}	41	V_{CC}	81	V_{CC}	121	V_{CC}
2	$\overline{\text{INTD}}$	42	AD14	82	LA23	122	LA6
3	$\overline{\text{PRST}}$	43	AD13	83	LD8/ID8	123	LD25/ID25
4	PCLK	44	AD12	84	LA22	124	LA5
5	$\overline{\text{GNT}}$	45	AD11	85	LD9/ID9	125	LD26/ID26
6	$\overline{\text{REQ}}$	46	AD10	86	LA21	126	LA4
7	AD31	47	AD9	87	LD10/ID10	127	LD27/ID27
8	AD30	48	AD8	88	LA20	128	LA3
9	AD29	49	C/$\overline{\text{BE0}}$	89	LD11/ID11	129	LD28/ID28
10	AD28	50	V_{CC}	90	LA19	130	LA2
11	GND	51	GND	91	LD12/ID12	131	LD29/ID29
12	AD27	52	AD7	92	LA18	132	LD30/ID30
13	AD26	53	AD6	93	LD13/ID13	133	LD31/ID31
14	AD25	54	AD5	94	LA17	134	'1' ’0’[a]
15	AD24	55	AD4	95	LD14/ID14	135	$\overline{\text{BTERM}}$ $\overline{\text{ERR}}$[a]
16	C/$\overline{\text{BE3}}$	56	AD3	96	LA16	136	$\overline{\text{READY}}$ $\overline{\text{RDY}}$[a]
17	IDSEL	57	AD2	97	LD15/ID15	137	HOLD $\overline{\text{LBREQ}}$[a]
18	AD23	58	AD1	98	LA15	138	HOLDA $\overline{\text{LBGNT}}$[a]
19	AD22	59	AD0	99	LD16/ID16	139	$\overline{\text{ADS}}$ $\overline{\text{LREQ}}$[a]
20	V_{CC}	60	V_{CC}	100	V_{CC}	140	V_{CC}
21	GND	61	GND	101	GND	141	GND
22	AD21	62	LD0/ID0	102	LA14	142	LCLK MEMCLK[a]

Table 5: Pin Assignments (cont'd)

PIN #	Signal	PIN #	Signal	PIN #	Signal	PIN #	Signal
23	AD20	63	LA31	103	LD17/ID17	143	$\overline{\text{EN5V}}$
24	AD19	64	LD1/ID1	104	LA13	144	V_{CC}
25	AD18	65	LA30	105	LD18/ID18	145	$\overline{\text{BE3}}$ $\overline{\text{BWE3}}$[a]
26	AD17	66	LD2/ID2	106	LA12	146	$\overline{\text{BE2}}$ $\overline{\text{BWE2}}$[a]
27	AD16	67	LA29	107	LD19/ID19	147	$\overline{\text{BE1}}$ $\overline{\text{BWE1}}$[a]
28	C/$\overline{\text{BE2}}$	68	LD3/ID3	108	LA11	148	$\overline{\text{BE0}}$ $\overline{\text{BWE0}}$[a]
29	$\overline{\text{FRAME}}$	69	LA28	109	LD20/ID20	149	$\overline{\text{BLAST}}$ $\overline{\text{BURST}}$[a]
30	GND	70	LD4/ID4	110	LA10	150	W/$\overline{\text{R}}$ R/$\overline{\text{W}}$[a]
31	$\overline{\text{IRDY}}$	71	LA27	111	LD21/ID21	151	RDIR
32	$\overline{\text{TRDY}}$	72	LD5/ID5	112	LA9	152	$\overline{\text{LRST}}$
33	$\overline{\text{DEVSEL}}$	73	LA26	113	LD22/ID22	153	'1'
34	$\overline{\text{STOP}}$	74	LD6/ID6	114	LA8	154	$\overline{\text{LINT}}$
35	$\overline{\text{PERR}}$	75	LA25	115	LD23/ID23	155	SDA
36	$\overline{\text{SERR}}$	76	LD7/ID7	116	LA7	156	SCL/ $\overline{\text{LPERR}}$
37	PAR	77	LA24	117	LPAR2	157	$\overline{\text{INTA}}$
38	C/$\overline{\text{BE1}}$	78	LPAR0	118	LPAR3	158	$\overline{\text{INTB}}$
39	AD15	79	LPAR1	119	LD24/ID24	159	$\overline{\text{INTC}}$
40	GND	80	GND	120	GND	160	GND

[a] Applies to AMD29030/40 mode.

Figure 1: Pinout for 160-pin EIAJ PQFP (top view)

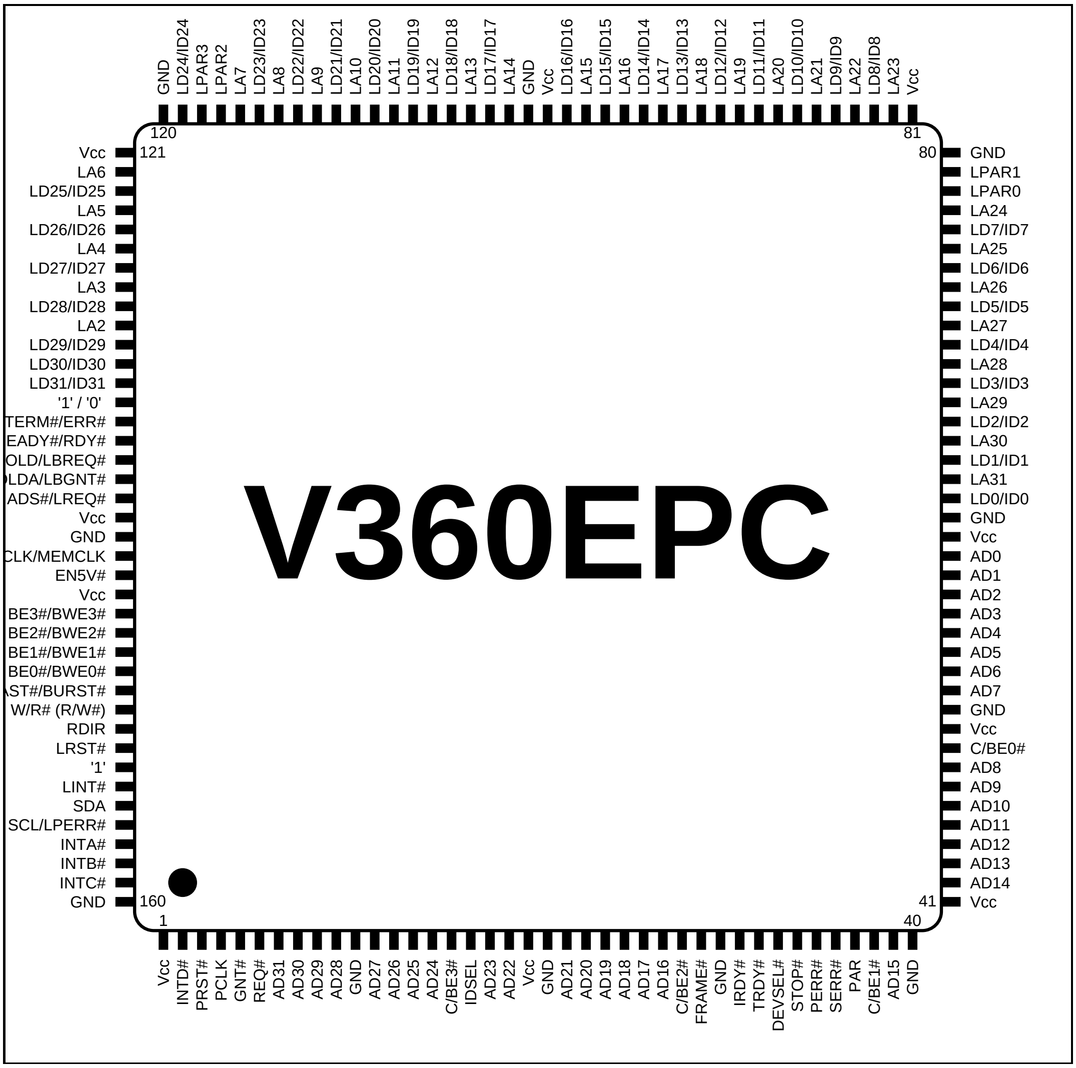
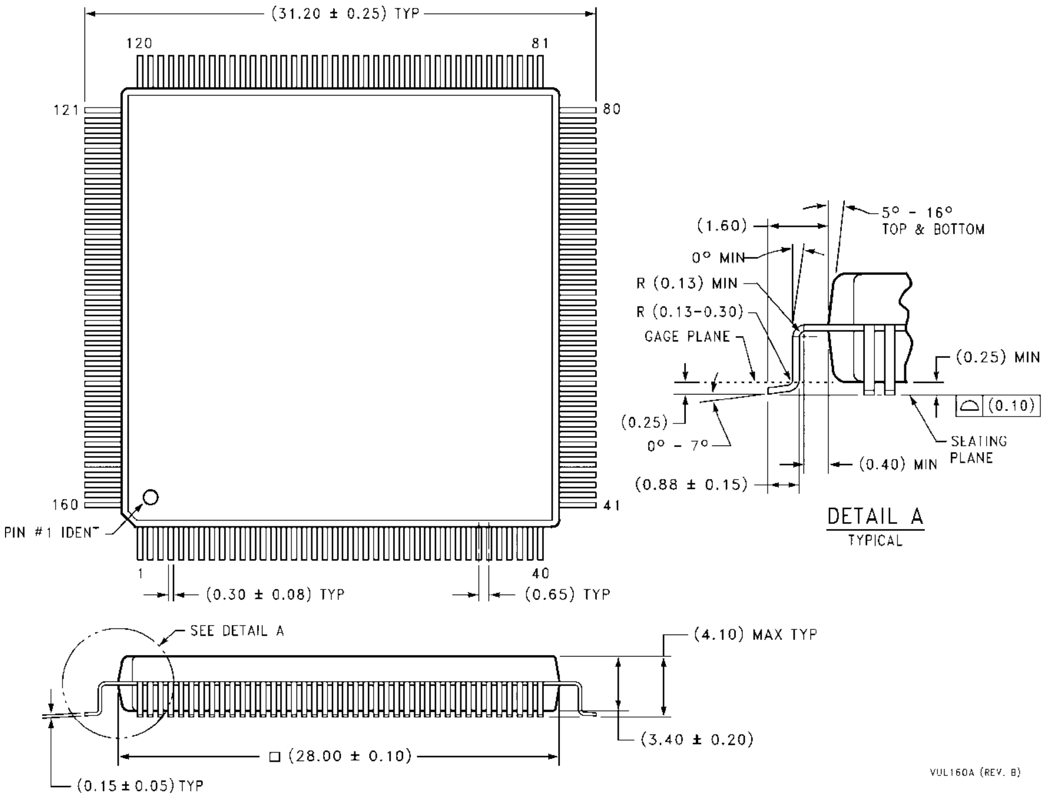


Figure 2: 160-pin EIAJ PQFP mechanical details



Unit of Measurement = millimeters

3.0 DC Specifications

The DC specifications for the PCI bus signals match exactly those given in the PCI Specification, Rev. 2.1, Section 4.2.1.1. For more information on the PCI DC specifications, see the PCI Specification.

Table 6: Absolute Maximum Ratings

Symbol	Parameter	Value	Units
V_{CC}	Supply voltage	-0.3 to +7	V
V_{IN}	DC input voltage	-0.3 to V_{CC}+0.3	V
I_{IN}	DC input current	± 10	mA
T_{STG}	Storage temperature range	-40 to +125	°C

Table 7: Guaranteed Operating Conditions

Symbol	Parameter	Value	Units
V_{CC}	Supply voltage	4.50 to 5.50	V
T_A	Ambient temperature range	-40 to 85	°C

3.1 PCI Bus DC Specifications

Table 8: PCI Bus Signals DC Operating Specifications

Symbol	Parameter	Condition	Min	Max	Units	Notes
V_{IH}	Input high voltage		2.0	V_{CC}+0.5	V	
V_{IL}	Input low voltage		-0.5	0.8	V	
I_{IH}	Input high leakage current	V_{IN} = 2.7V		70	µA	1
I_{IL}	Input low leakage current	V_{IN} = 0.5V		-70	µA	1
V_{OH}	Output high voltage	I_{OUT} = -2mA	2.4		V	
V_{OL}	Output low voltage	I_{OUT} = 3mA, 6mA		0.55	V	2
C_{IN}	Input pin capacitance			10	pF	3
C_{CLK}	PCLK pin capacitance		5	12	pF	
C_{IDSEL}	IDSEL pin capacitance			8	pF	4
L_{PIN}	Pin inductance			20	nH	

Notes:
1. Input leakage currents include high impedance output leakage for all bi-directional buffers with tri-state outputs.
2. Signals without pullup resistors have greater than 3mA low output current. Signals requiring pull resistors have greater than 6mA output current. The latter include $\overline{\text{FRAME}}$, $\overline{\text{TRDY}}$, $\overline{\text{IRDY}}$, $\overline{\text{STOP}}$, $\overline{\text{SERR}}$, $\overline{\text{PERR}}$.
3. Absolute maximum pin capacitance for a PCI unit is 10pF (except for CLK).
4. Lower capacitance on this input-only pin allows for non-resistive coupling to AD[xx].

3.2 Local Bus DC Specifications

Table 9: Local Bus Signals DC Operating Specifications

Symbol	Description	Conditions	Min	Max	Units
V_{IL}	Low level input voltage	V_{CC} = 4.75V		0.8	V
V_{IH}	High level input voltage	V_{CC} = 5.25V	2.0		V
I_{IL}	Low level input current	V_{IN}=GND, V_{CC}=5.25V	-10		µA
I_{IH}	High level input current	V_{IN} = V_{CC} = 5.25V		10	µA
V_{OL4}	Low level output voltage for 4 mA outputs and I/O pins	I_{OL} = -4 mA		0.4	V
V_{OH4}	High level output voltage for 4 mA outputs and I/O pins	I_{OH} = 4 mA	2.4		V
I_{OZL}	Low level float input leakage	V_{IN} = GND	-10		µA
I_{OZH}	High level float input leakage	V_{IN} = V_{CC}		10	µA
I_{CC} (max)	Maximum supply current	V_{CC} = 5.25V PCLK = LCLK = 33MHz		150	mA
I_{CC} (typ)	Typical supply current	V_{CC} = 5.0V PCLK = LCLK = 33MHz		120	mA
C_{IO}	Input and output capacitance			10	pF

4.0 AC Specifications

The AC specifications for the PCI bus signals match exactly those given in the PCI Specification, Rev. 2.1, Section 4.2.1.2. For more information on the PCI AC specifications, including the V/I curves for 5V signalling, see section 4.2.1.2 of Rev 2.1 PCI Specification.

4.1 PCI Bus Timings

Table 10: PCI Bus Signals AC Operating Specifications

Symbol	Parameter	Condition	Min	Max	Units	Notes
$I_{OH(AC)}$	Switching current high	0V<V_{OUT}≤1.4V	-44		mA	1
		1.4V<V_{OUT}<2.4V	-44+(V_{OUT}-1.4)/0.024	Equation A	mA	1, 2, 3
	(Test point)	V_{OUT}=3.1V		-142	mA	3
$I_{OL(AC)}$	Switching current low	V_{OUT}≥2.2V	95		mA	1
		2.2V>V_{OUT}>0.55	V_{OUT}/0.023	Equation B	mA	1, 3
	(Test point)	V_{OUT}=0.71		206	mA	3
I_{CL}	Low clamp current	-5<V_{IN}≤-1	-25+(V_{IN}+1)/0.015		mA	
t_R	Unloaded output rise time	0.4V to 2.4V	1	5	V/ns	4
t_F	Unloaded output fall time	2.4V to 0.4V	1	5	V/ns	4

Notes:
1. Refer to the V/I curves in Section 4.2.1 of the PCI Specification. This specification does not apply to CLK and $\overline{\text{RST}}$ which are system outputs. “Switching Current High” specifications are not relevant to open drain outputs such as $\overline{\text{SERR}}$ and $\overline{\text{INTA}}$-$\overline{\text{INTD}}$.
2. Note that this segment of the minimum current curve is drawn from the AC drive point directly to the DC drive point rather than toward the voltage rail (as it does in the pull-down curve). This difference is intended to allow for an optional N-channel pullup.
3. Maximum current requirements are met as drivers pull beyond the first step voltage (AC drive point). Equations defining these maximums (A and B) are provided with the respective V/I curves given in the PCI Spec. The equation defined maxima is met by design.
4. The minimum slew rate (slowest signal edge) is met by the PCI drivers. The maximum slew rate (fastest signal edge) is a guideline. Motherboard designers must bear in mind that rise and fall times faster than this maximum guideline could occur, and should ensure that signal integrity modeling accounts for this.

Equation A: $I_{OH} = 11.9\cdot(V_{OUT} - 5.25V)\cdot(V_{OUT} + 2.45V)$ for $V_{CC} > V_{OUT} > 3.1V$

Equation B: $I_{OL} = 78.5\cdot V_{OUT}(4.4V - V_{OUT})$ for $0V < V_{OUT} < 0.71V$

4.2 Local Bus Timings

Table 11: Local Bus AC Test Conditions

Symbol	Parameter	Limits	Units
V_{CC}	Supply voltage	4.50 to 5.50	V
V_{IN}	Input low and high voltages	0.4 and 2.0	V
C_{OUT}	Capacitive load on output and I/O pins	50	pF

Table 12: Capacitive Derating for Output and I/O Pins

Output Drive Limit	Derating
4mA	0.058 ns/pF for loads > 50pF

Figure 3: Clock and Synchronous Signals

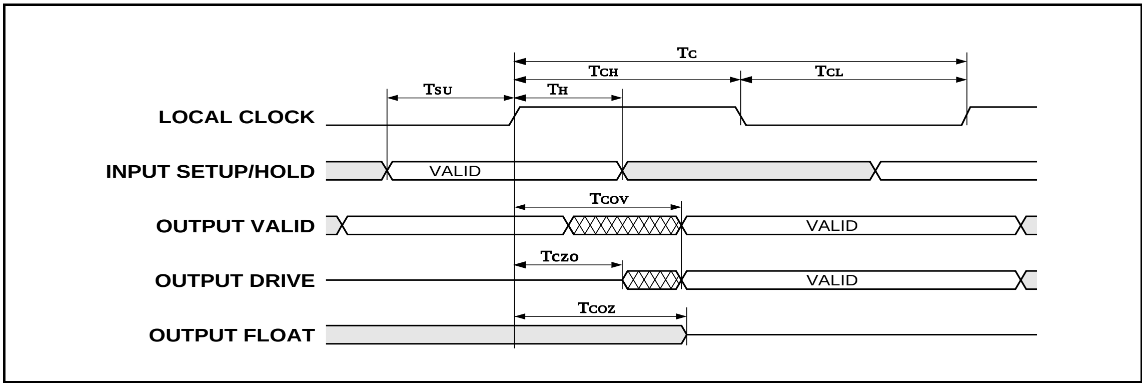


Table 13: Local Bus Timing Parameters for Vcc = 5 Volts +/- 5%

				33MHz		50MHz		
#	Symbol	Description	Notes	Min	Max	Min	Max	Units
1	T_C	LCLK/MEMCLK period		30		20		ns
2	T_{CH}	LCLK/MEMCLK high time	1	12		9		ns
3	T_{CL}	LCLK/MEMCLK low time	1	12		9		ns
4	T_{SU}	Synchronous input setup	2	7		6		ns
4a	T_{SU}	Synchronous input setup ($\overline{\text{BLAST}}$,$\overline{\text{BTERM}}$)/($\overline{\text{BURST}}$, $\overline{\text{ERR}}$)		8		7		ns
4b	T_{SU}	Synchronous input setup ($\overline{\text{ADS}}$/$\overline{\text{LREQ}}$)		6		5		ns
4c	T_{SU}	Synchronous input setup (address, data, byte enables)		8		6		ns
4d	T_{SU}	Synchronous input setup for read data when in local bus master mode		5		5		ns
4e	T_{SU}	Synchronous input setup for ($\overline{\text{READY}}$, W/$\overline{\text{R}}$, HOLDA)/($\overline{\text{RDY}}$, R/$\overline{\text{W}}$, $\overline{\text{LBGRT}}$)		5		4		
5	T_H	Synchronous input hold			2		2	ns
6	T_{COV}	LCLK/MEMCLK to output valid delay	3	3	14	3	10	ns
6a	T_{COV}	LCLK/MEMCLK to output valid delay (address, data, byte enable, parity)		3	15	3	12	ns
7	T_{CZO}	LCLK to output driving delay		3	15	3	12	ns
8	T_{COZ}	LCLK/MEMCLK to high impedance delay	4	3	15	3	12	ns
9	T_{RST}	Reset period when LRST used as input		$16 \cdot T_C$		$16 \cdot T_C$		ns

Notes:
1. Measured at 1.5V.
2. All local bus signals except those in 4a, 4b, 4c, 4d and 4e.
3. All local bus signals except those in 6a.
4. $\overline{\text{READY}}$, $\overline{\text{BLAST}}$, $\overline{\text{ADS}}$ are driven to high impedance at the falling edge of LCLK.

Table 14: PCI Bus Timing Parameters for Vcc = 5 Volts +/- 10%

#	Symbol	Description	Notes	Min	Max	Units
1	T_C	PCLK period		30		ns
2	T_{SU}	Synchronous input setup to PCLK	1	7		ns

Table 14: PCI Bus Timing Parameters for Vcc = 5 Volts +/- 10%

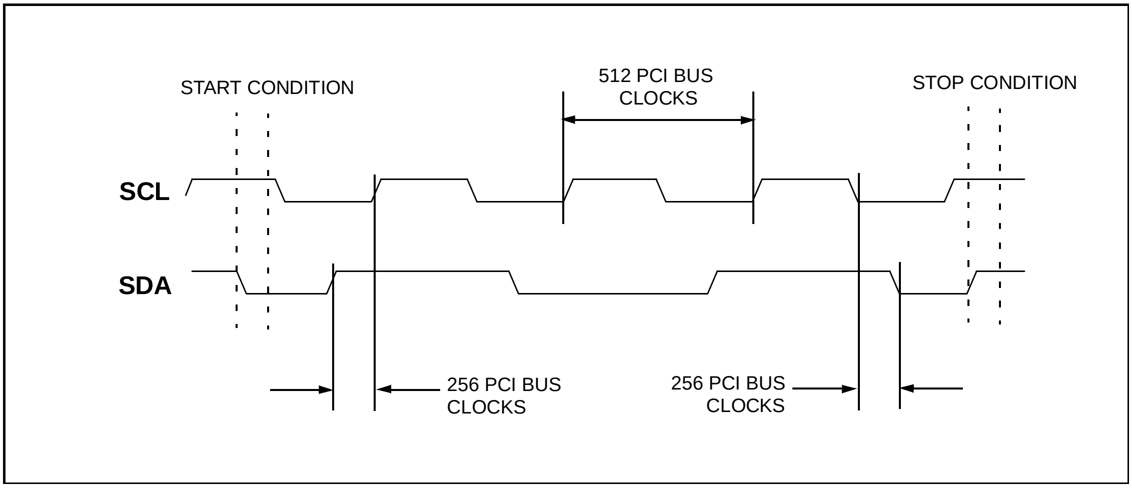
2a	T_{SU}	Synchronous input setup to PCLK ($\overline{\text{GNT}}$)		10		ns
3	T_H	Synchronous input hold from PCLK		0		ns
4	T_{COV}	PCLK to output valid delay	2	3	11	ns
4a	T_{COV}	PCLK to output valid delay ($\overline{\text{REQ}}$)		4	12	ns
5	T_{CZO}	PCLK to output driving delay		4	11	ns
6	T_{COZ}	PCLK to high impedance delay		5	18	ns
7	T_{RST}	Reset period when PRST used as input		$16 \cdot T_C$		

Notes:
1. All PCI bus signals except those in 2a.
2. All PCI bus signals except those in 4a.

4.3 Serial EEPROM Port TImings

The clock for the serial EEPROM interface is derived by dividing the PCI bus clock. The waveforms generated are shown in Figure 4.

Figure 4: Serial EEPROM Waveforms and Timings



5.0 Revision History

Table 15: Revision History

Revision Number	Date	Comments and Changes
1.0	8/97	First pre-silicon revision of preliminary data sheet.

SEMICONDUCTOR
THE EMBEDDED CHIPSET COMPANY

USA:
2348G Walsh Ave.
Santa Clara CA 95051
Phone: (408)988-1050 Fax: (408)988-2601
Toll Free: (800)488-8410 (Canada and U.S. only)
World Wide Web: http://www.vcubed.com

Temporary
EPC
Users Manual

The EPC Users Manual
is currently being re-written.

V350EPC, V360EPC

Local Bus to PCI Bridge

User's Manual

Revision 0.2x

V3 Semiconductor makes no warranties for the use of its products. V3 does not assume any liability for errors which may appear in this document, however, we will attempt to notify customers of such errors.

V3 Semiconductor retains the right to make changes to either the documentation, specification or component without notice.
Please verify with V3 Semiconductor to be sure you have the latest specifications before finalizing a design.

Contents

Chapter 1 Introduction

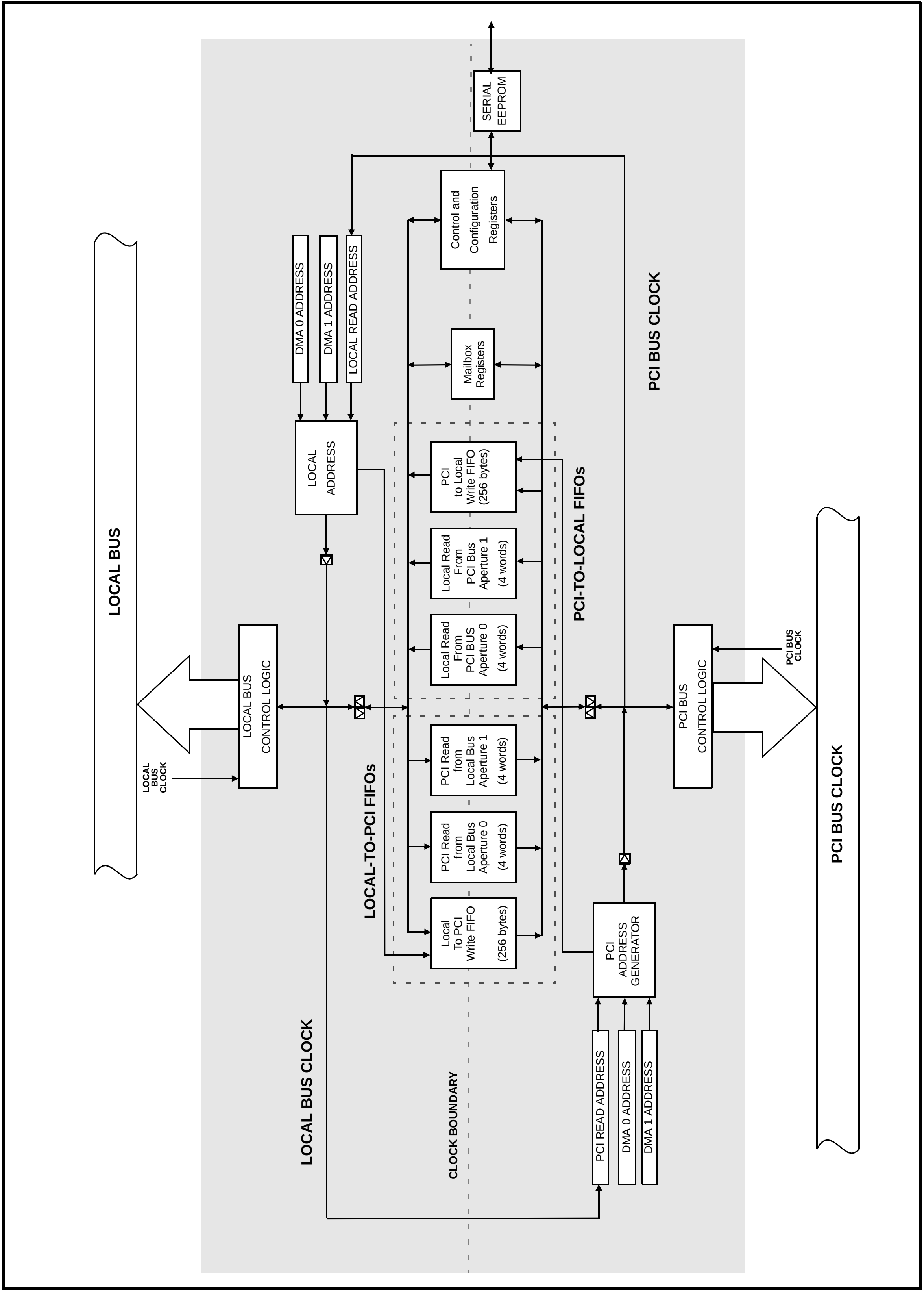
In a very short period of time the PCI bus standard has moved beyond the PC to become the most widely accepted high-performance bus standard for embedded applications. As a leader in providing chipset solutions for high-end embedded applications, V3 Semiconductor has developed the EPC family of PCI Bridge Components for the Intel i960® and the AMD Am29K™ processor family. The EPC is specifically designed to take advantage of the key features of the i960/Am29K family of processors to deliver the highest performance possible for embedded PCI systems.

The EPC is the new enhanced version of the previous generation of PCI Bridges known as the PBC. The V350EPC is backward compatible (register and pin) with the V960PBC and the V961PBC devices. The V360EPC is backward compatible (register and pin) with the V962PBC and the V292PBC. A block diagram of the EPC is shown in Figure 1.

Some of the key features of the EPC are:

- Glueless interface to i960/Am29K processors
- Compliant with PCI 2.1 specification, PCI Hot Swap Support
- Configurable for system host, bus master, and target operation
- Burst access support on both local and PCI interfaces
- PCI-to-Local and Local-to-PCI address space remapping
- 2 PCI-to-Local and 2 Local-to-PCI data transfer apertures
- 640 bytes of programmable FIFO storage with *DYNAMIC BANDWIDTH* ALLOCATION™
- On-the-fly byte order (endian) conversion
- 2 channel DMA controller with DMA Chaining
- Bi-directional mailbox registers with doorbell interrupts
- Power on configuration via serial EEPROM
- Direct connect to VxBMC/CMC or V96SSC memory controllers

Figure 1: EPC BLOCK DIAGRAM



1.1 HOW TO USE THIS MANUAL

The EPC User's Manual includes detailed information regarding the programming and design of systems based on the EPC. However there are other complementary documents to assist designers. A complete set of documentation includes the following:

V350EPC

- EPC User's Manual (this document)
- V350EPC Data Sheet
- Reference design manual (Quatro, Avalanche)

V360EPC

- EPC User's Manual (this document)
- V360EPC Data Sheet
- Reference design manual (Quatro, Avalanche)

The electrical specifications are found in the V350EPC, V360EPC Data Sheets. Before finalizing a system design based on the EPC, please contact V3 Semiconductor to verify that you have the most recent specifications. Other application notes and useful design aids can be found on the V3 Web site.

V3 is constantly trying to improve the quality of its product documentation. If you have any questions or comments, please contact V3 Customer Assistance.

1.2 GETTING HELP FROM V3 SEMICONDUCTOR

If you need assistance with a technical question, please contact V3 through our AppsFax or Email hotlines. Email is the quickest and most efficient way to get technical support from V3.

If you do not have access to Email or a fax machine, feel free to call us from 9AM to 5PM Pacific Standard Time.

V3 AppsFax:	(408) 988-2601 (Santa Clara, California)
V3 Customer Assistance:	(800) 488-8410 (US and Canada)
	(408) 988-1050 (Outside North America)
	v3help@vcubed.com

Some technical support information is also posted on the V3 Web site. This is the source of the most up-to-date data sheets and user's manuals and is located at:

www.vcubed.com

1.3 GETTING ANSWERS TO PCI RELATED QUESTIONS

This manual assumes a basic understanding of the PCI bus specification. If you are looking for a copy of the specification please contact the PCI special interest group at 800.433.5177.

If you are not intimately familiar with the PCI specification, a good place to start is by reading one of several books on the subject. One of the most popular is "PCI System Architecture" written by Tom Shanley and Don Anderson (published by MindShare Inc.).

1.4 GETTING INFORMATION ABOUT THE i960/Am29K FAMILY

This manual assumes that you are familiar with the i960/Am29K processors for which the EPC was designed. For information about products, you may call Intel / Advanced Micro Devices at the numbers listed below.

Intel:	(800) 879-4683 (US and Canada)
	(408) 987-8080 (Outside North America)
	www.intel.com (Web site)
AMD:	(800) 222-9323 (US and Canada)
	(408) 749-5703 (Outside North America)
	www.amd.com (Web site)

1.5 DISCLAIMER

V3 Semiconductor makes no warranties for the use of its products. V3 does not assume any liability for errors which may appear in this document, however, we will attempt to notify customers of such errors. V3 Semiconductor retains the right to make changes to either the documentation, specification or component without notice.

Please verify with V3 Semiconductor to be sure you have the latest specifications before finalizing your design.

1.6 REVISION HISTORY

Table 1: Revision History

Revision Number	Date	Comments
0.2	2/98	First release without NDA.

Chapter 2 Bridge Operation Overview

The EPC supports four modes of PCI operation:

- ***System Master*** - The i960/Am29K processor is the PCI system master and PCI broadcasts configuration commands to attached PCI subsystems. This mode would be used, for example, in embedded systems using PCI as the system mezzanine bus.

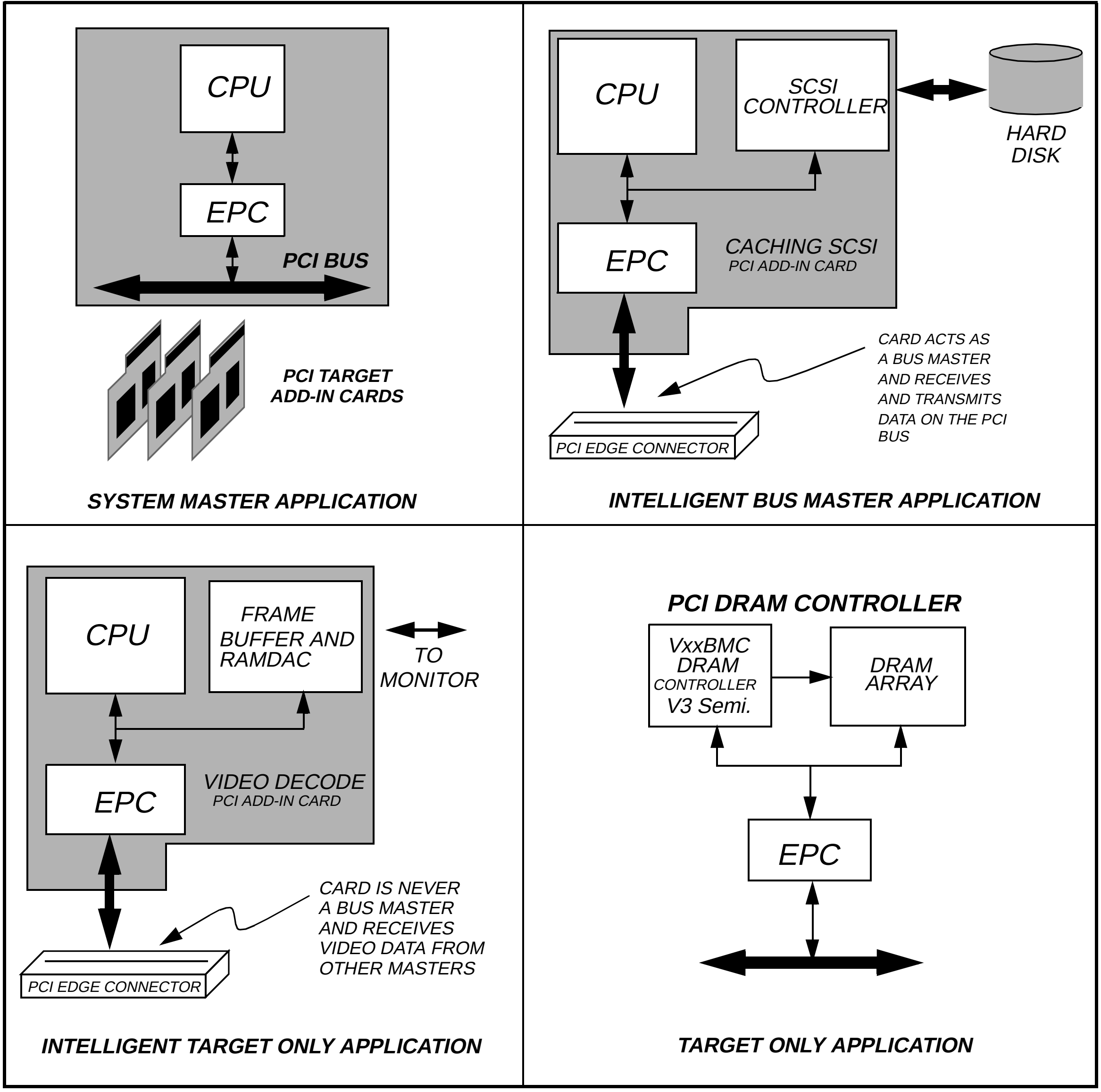
- ***Bus Master*** **-** The i960/Am29K processor is part of a PCI subsystem that receives configuration commands from a primary system master, yet will act as a bus master during PCI transactions. This mode would be used, for example, for intelligent PCI add-in cards based on the i960/Am29K processor.

- ***Target*** **-** The i960/Am29K processor is part of a PCI subsystem that receives configuration commands from a host CPU and does not act as a PCI bus master.

- ***Stand Alone Target*** - The EPC can be used as a stand alone bridge component without an attached processor. For example, the EPC can be used with the VxBMC/CMC or V96SSC Burst DRAM Controller to build a two-chip PCI-to-DRAM interface.

Figure 2 shows example designs covering each of the four modes of operation.

Figure 2: Example EPC System Designs



2.1 OPERATIONAL EXAMPLE

The easiest way to understand the operation of the EPC is to cover several simple operational examples. In this section we will briefly describe: PCI-to-Local transfers, Local-to-PCI transfers, DMA operation, and mailbox register usage. Initialization and configuration is deferred to later chapters.

After initialization the EPC monitors both the local and PCI buses simultaneously, waiting for access within pre-programmed regions of local memory or PCI space. Two independent programmable apertures are provided for local memory to PCI bus transfers; two more apertures are provided for PCI to local memory transfers. The apertures can also perform address translation and byte-order conversion on accesses flowing across the bridge.

For our operational example, let’s assume the following:

- The base address for PCI-to-Local aperture 0 is set at 1000.0000H with a size of 1 megabyte
- The base address for Local-to-PCI aperture 0 is set at E000.0000H with a size of 4 megabytes
- PCI-to-Local aperture 1 is disabled
- Local-to-PCI aperture 1 is disabled
- No errors occur during the transfers

2.1.1 Direct Local Bus Write to PCI Space

When the local bus master needs to perform a write to a device on the PCI bus, it simply writes data to a local memory location within a Local-to-PCI aperture. In this example, all local bus writes with an address within the range E000.0000H to E03F.FFFFH (base at E000.0000H with 4 megabyte window), will be "captured" and converted to PCI bus transfers. To the local bus master it appears that the write has completed. However, the address and data for the write have been placed into the Local-to-PCI bus write FIFO for completion when the PCI bus becomes available.

Both the address and data for the transfer can be changed as they flow across the bridge. If address translation is selected, the target address for the write can be changed from the local bus address to a different address in PCI space. For example, the bridge can be programmed to convert a write to location E000.0020H in local space, into a write to location A000.0020H in PCI space. In addition, the data for the write may optionally be converted from big endian to little endian byte order or vice-versa. Byte order conversion is useful in systems where the local processing uses one byte order and the main CPU uses another byte order. A good example is a networking PC add-in card that processes data in big-endian order, but has to share that data with an x86 processor host, which expects little endian byte order.

Once the "captured" write access reaches the PCI side of the FIFO, the EPC requests ownership of the PCI bus (how long the EPC waits before requesting the bus is programmable). When the central PCI arbiter grants the EPC the bus, the EPC then bursts the data queued in the write FIFO to the target PCI address. The EPC relinquishes ownership of the PCI bus once the write transfer is completed.

2.1.2 Direct Local Bus Read from PCI Space

A local bus read from PCI space progresses similarly to the write example above. When the local bus reads from a location in the Local-to-PCI aperture (in this case E000.0000H to E03F.FFFFH), these reads are converted into read transfers on the PCI bus. Unlike the write case, however, the local master cannot complete its read operation until the data is delivered from PCI space.

For example, let’s look at the case where the local master wants to read from a register on an add-in card located at A000.0010H in PCI space. Let’s assume the EPC is programmed to translate accesses to Local-to-PCI aperture 0 from E00n.nnnn in local memory to A00n.nnnnH in PCI space. To read from A000.0010H, the local master initiates a read to E000.0010H. The EPC sees this read, recognizes it as being within Local-to-PCI aperture 0 and "captures" it for bridging to the PCI bus. Since the data is not immediately available to return to the host, the bridge returns a NOT READY indication to the local processor. Simultaneously, the EPC requests the PCI bus. Once granted the bus, the EPC begins to read from the translated address (A000.0010H) and transfers that data back across the bridge to the local processor. As each datum becomes available on the local side, READY is returned to complete the local bus request. Just as in the write transfer case, the EPC can be programmed to perform byte order conversion as the data flows through the bridge. In order to improve system bandwidth, the EPC can be programmed to prefetch data from the PCI bus. The prefetched data will be ‘cached’ in the bridge until the next access.

2.1.3 PCI Write to Local Space

PCI bus masters gain access to the local memory space through the PCI-to-Local bus apertures. In our example, the PCI-to-Local aperture has been programmed to respond to PCI accesses within the 1000.0000H to 100F.FFFFH range (a 1 megabyte window). For a PCI master to write into the EPC’s local memory, it simply writes to a PCI location falling within a PCI-to-Local aperture window.

For example, let’s assume the PCI master wants to write to location 2000.0030H in the EPC’s local memory space. Since the bridge is programmed to respond to accesses in the 1000.0000H to 100F.FFFFH range we will need to re-map the address so that the proper translation occurs. The PCI master now writes data to PCI location 1000.0030H, that access is captured by the bridge and buffered within the PCI-to-Local write FIFO. The PCI master completes the write and relinquishes the bus. Simultaneously, the address of the access is translated within the bridge, and the EPC requests access to the local bus (the protocol used to gain local bus mastership is processor version dependent). Once granted the local bus, the EPC will write the captured data in the new location in local memory space (2000.0030H). Data byte order translation is also available in the PCI-to-Local direction.

2.1.4 PCI Reads from Local Space

As you might expect, PCI reads from local space closely mirror local reads from PCI space. Let’s assume the same conditions as the previous example, except in this case the PCI master wants to read from location 2000.0030H in local memory space. In this case, the PCI master reads from location 1000.0030H in PCI space (remember address translation will do this automatically once configured). The bridge immediately responds by deasserting "target ready" ($\overline{\text{TRDY}}$) to inform the PCI master that it will need to wait for the data to be fetched from the local side. Simultaneously, the EPC requests the local bus. Once granted the local bus, the EPC will read from location 2000.0030H and forward that data across the bridge to the PCI side. When the data is valid on the PCI side, $\overline{\text{TRDY}}$ will be asserted to signal completion of the read. As with the Local-to-PCI transfer, the EPC can prefetch data and cache it.

2.1.5 DMA Transfers

All four of the above examples assumed that the local processor, or a PCI master, was involved in the data transfer across the bridge. Often, higher overall system performance can be achieved by allowing simple transfers to be handled by a less-intelligent agent, such as a DMA controller. The EPC provides 2 DMA controllers, each capable of transferring data across the bridge without processor intervention.

Let’s take the example of a caching disk controller PCI add-in card. The local processor is responsible for retrieving data from the hard drive and building buffers in local memory.

When the system host processor, a PowerPC processor for example, requests a specific buffer be moved into system memory, the local processor programs the bridge with the start address and transfer count for the data buffer, and the target address in PCI space for the data. The DMA controller then transfers this data autonomously, allowing both the local processor and the system processor to go about their business. The DMA controller can also be programmed to transfer a chain of buffers. This is useful when, in the above example, several 512 byte sectors must be assembled into a 16K byte logical block in the host’s memory.

2.1.6 Mailbox Registers and Doorbell Interrupts

Often it is not practical for the EPC to request access to the local or PCI bus to transfer small amounts of information. Let’s use the caching disk card as an example again. Perhaps the PowerPC system master wants to know if the disk access has been completed (i.e. the data has been moved from the local memory buffer into system memory). One way to do this would be to set up a semaphore in local memory that the local processor sets whenever a transfer was completed. To read the semaphore, the PowerPC system master would need to perform a cross-bridge read of local memory space, a potentially time consuming transfer to retrieve 1 bit of information.

A higher performance method to transfer small amounts of information is provided by the mailbox registers. These registers reside within the EPC and may be read or written from either side of the bridge. Each register is eight bits and the bit definition is application dependent. Using our caching disk controller example, one bit may be used to indicate "transfer complete". The local processor sets this bit by writing to a specific location within the EPC’s configuration space (independent of the transfer apertures). The system master processor would then read this register directly from the EPC; no local bus access is required.

In addition to transferring data, the mailbox registers can be used to generate interrupts on either side of the bridge. For example, you may use one register to indicate transfer data *NOW*, in which case you would want to immediately interrupt the local processor.

Chapter 3 Internal Register Apertures

The EPC does not use traditional chip-selects to access either its configuration registers or bridging functions. The chip-select function is replaced by address apertures, which compare addresses on both the PCI and Local bus. If an access is seen within one of these programmable apertures, then an internal chip-select is generated by the EPC.

There are a total of nine apertures implemented in the EPC:

- Four data transfer apertures that are used in the movement of data across the bridge. There are two apertures for data movement from PCI-to-Local and two for Local-to-PCI transfers. The data transfer apertures are described in detail in the next section.

- One PCI EPROM data transfer aperture is provided to allow location of a host system boot ROM on an adapter card. The boot ROM aperture can only transfer data from the local bus to PCI. Boot ROM support is discussed in the “PCI Configuration” chapter.

- One DOS Compatibility data transfer aperture is provided to allow real-mode DOS access to EPC local memory from the lower 1 megabyte of memory space and from DOS I/O holes. DOS memory and I/O support is discussed in the “PC Compatibility” chapter.

- One aperture is provided for access to the EPC’s internal registers from the Local bus.

- One aperture is provided for access to the EPC’s internal registers from the PCI bus (memory or IO cycles).

- One aperture is provided for access to the EPC’s internal registers from the PCI bus (PCI configuration cycles).

The function of the apertures is summarized in Figure 3.

3.1 LOCAL BUS ACCESS TO INTERNAL REGISTERS

Local bus access to internal registers is controlled by the Local-to-Internal Register aperture. The base address for this register is set in the LB_IO_BASE register. The LB_IO_BASE register is initialized by special bus cycles on the local bus immediately following a reset (see “Initialization”) or by serial EEPROM.

The LB_IO_BASE aperture has a fixed size of 64 kilobytes, although only a small part of this is actually used.

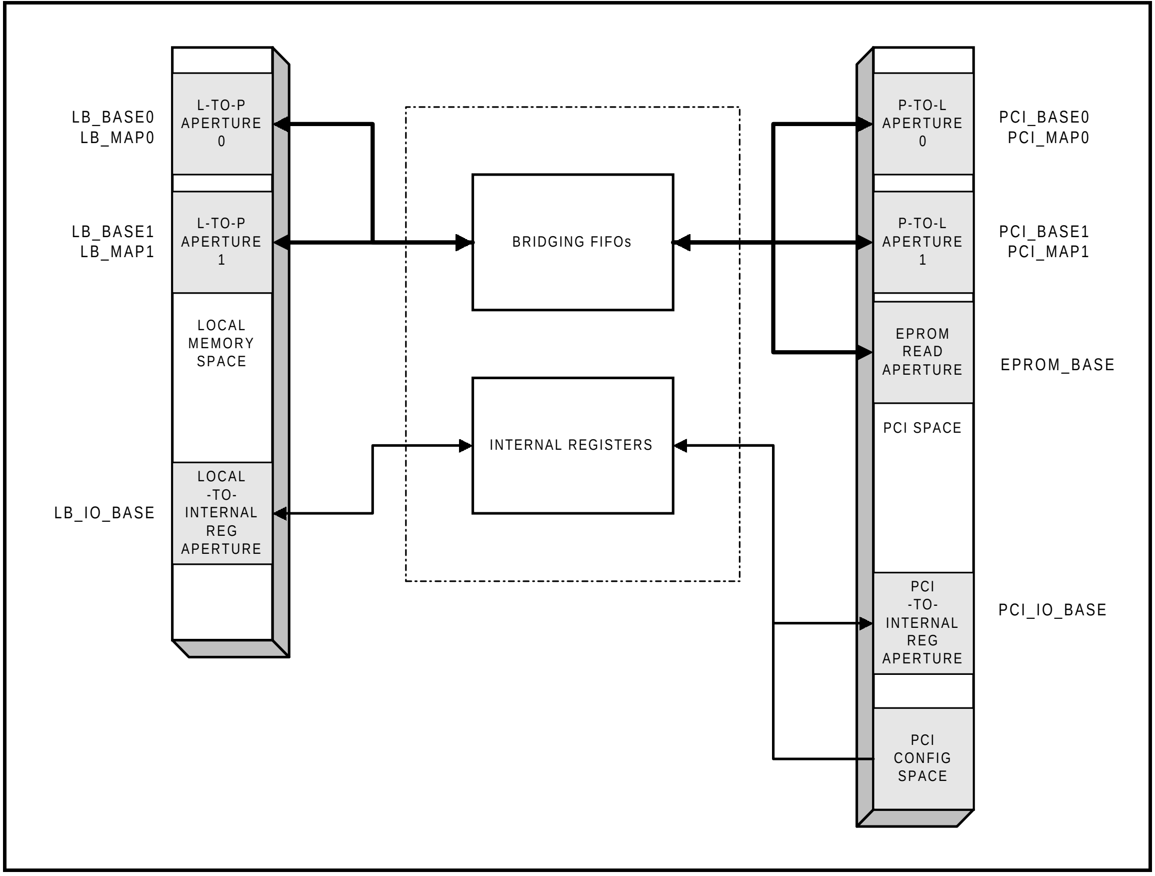
To write to a specific register through the LB_IO_BASE aperture, add the offset of the target register to the starting address of the Local-to-Internal Register aperture and use that address for the write. For example, if the LB_IO_BASE register is programmed to place the Local-to-Internal Register aperture at E123.0000H and you want to write to the PCI Command Register (offset 4H), then the target address would be E123.0004H.

3.2 PCI BUS ACCESS TO INTERNAL REGISTERS

PCI bus access to internal registers is controlled by the PCI-to-Internal Register aperture. The base address for this register is set in the PCI_IO_BASE register. The PCI_IO_BASE register is initialized either by the serial EEPROM, PCI configuration cycles, or local bus register accesses (see “Initialization”). The PCI_IO_BASE aperture has a fixed size of 256 bytes and may be located in either PCI memory or IO space.

The EPC’s internal registers can also be accessed by standard PCI configuration cycles. Configuration cycles are initiated by a PCI system master by driving active the EPC’s IDSEL pin, and then performing configuration reads and writes. For configuration reads/writes, only the low 8 bits of the address are used to index the internal registers. AD[31:8] are ignored during the address phase.

Figure 3: Summary of Aperture Function



Chapter 4 Data Transfer Apertures

There are four apertures provided for cross-bridge data transfers:

- PCI-to-Local apertures 0 and 1
- Local-to-PCI apertures 0 and 1

These apertures are tightly coupled to the read and write FIFOs, as well as to the address translation and data order conversion logic as shown in Figure 4.

Each aperture includes an address comparator that sets the base and size of the aperture. Accesses recognized by the address comparator are forwarded through the address remapper, byte order converter, and FIFO for that particular aperture.

Additional data transfer apertures are provided for PC compatibility. These apertures are discussed in the “PC Compatibility” chapter.

4.1 PCI-TO-LOCAL BUS APERTURES

The PCI-to-Local bus apertures control the following accesses: writes from PCI to Local memory and reads from local memory destined for PCI space. The PCI-to-Local bus apertures are implemented using the standard *Base Register* formats in the PCI configuration header. Unused base registers return all zeros when read or interrogated during configuration (see "PCI Configuration").

The programming of the PCI-to-Local apertures is controlled via the PCI_BASEx and PCI_MAPx registers. The following options are programmable for each aperture:

- Base address of aperture
- Aperture size
- Type of PCI accesses to respond to (IO or memory)
- Address translation
- Data byte ordering conversion
- Read prefetch enable/disable

4.1.1 Setting the PCI-to-Local Aperture Base Address and Size

The base address for a PCI-to-Local memory aperture is set in the ADR_BASE field of the PCI_BASEx register (see "Register Descriptions" chapter for register layouts). In non-DOS mode, only AD[31:20] are significant yielding a minimum base address granularity of 1Mbyte.

The size of a PCI-to-Local aperture is set via the ADR_SIZE field in the PCI_MAPx register. Supported sizes for memory access are from 1 megabyte to 2 gigabytes increasing as powers-of-2 (1M, 2M, 4M, etc.)

When an aperture size greater than 1 megabyte is selected, the corresponding bits in the MAP_ADR field of the PCI_BASEx register are ignored. For example, if you choose an aperture size of 8 megabytes, then address bits A20, A21 and A22 are "masked off" by the PCI-side aperture address comparators.

In DOS Mode, the address comparators function differently allowing greater granularity for I/O and memory accesses (down to 8 bytes). Please see the chapter titled "DOS Compatibility" for more details.

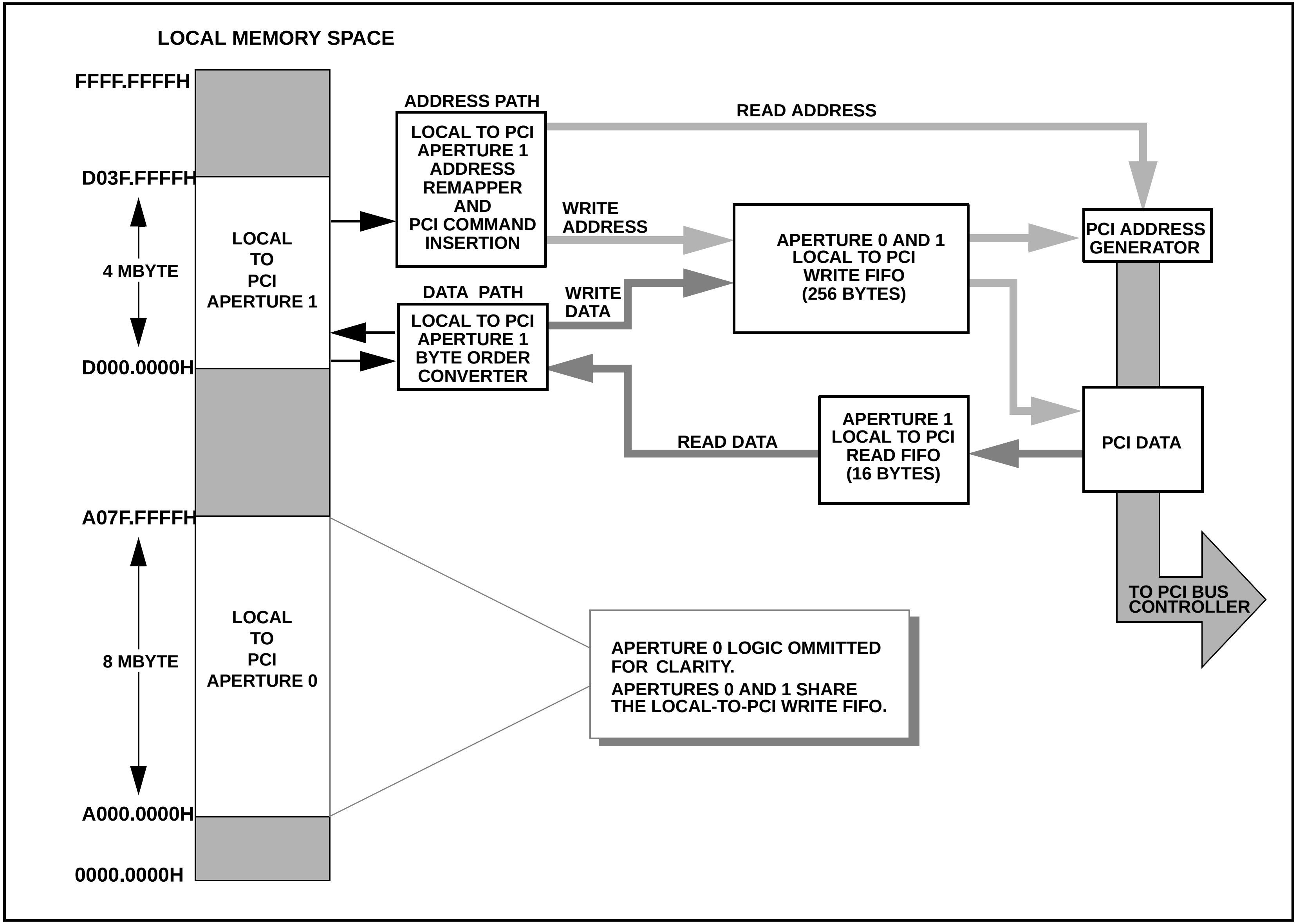
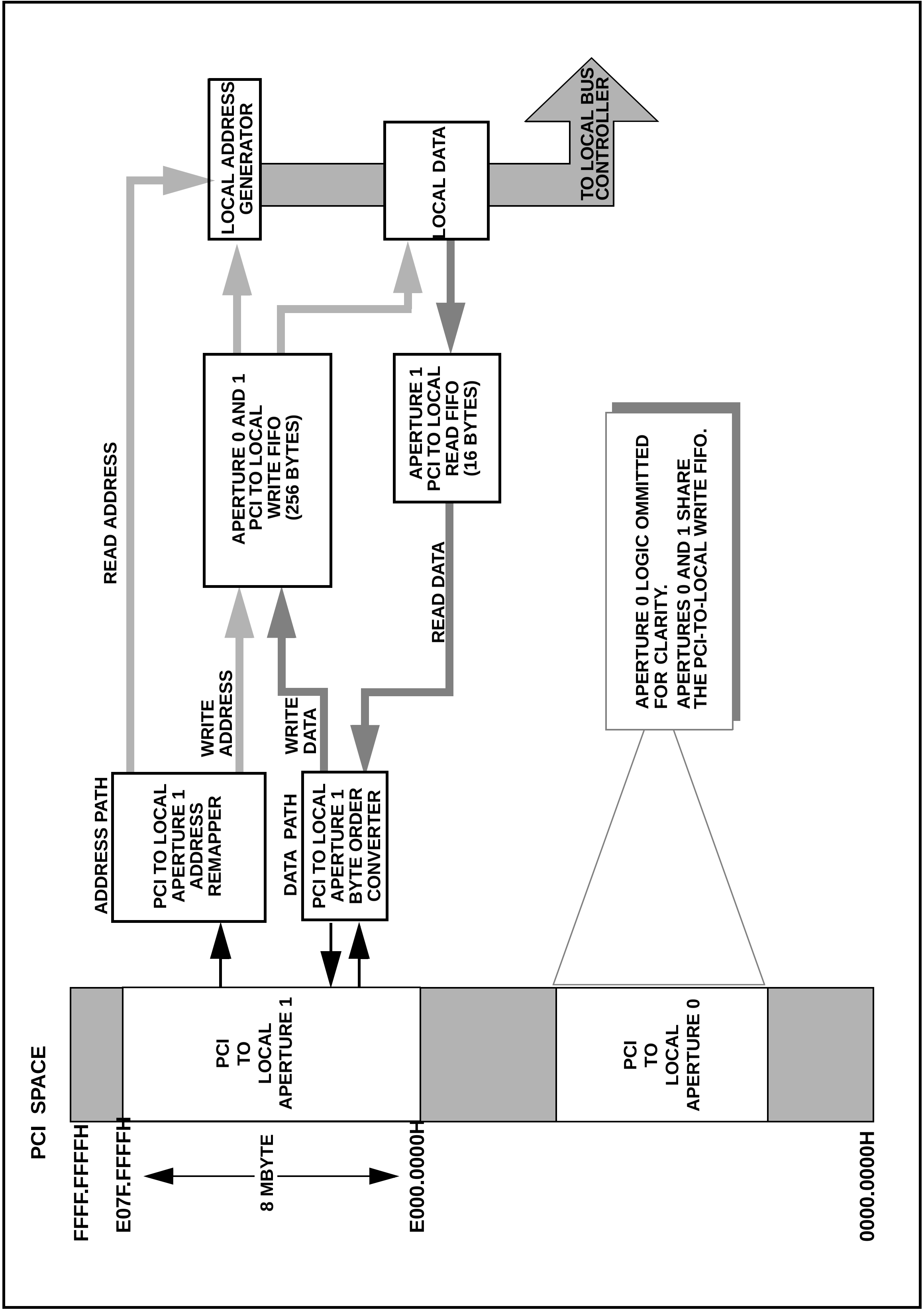

LOCAL MEMORY SPACE
FFFF.FFFFH
D03F.FFFFH
4 MBYTE
LOCAL TO PCI APERTURE 1
D000.0000H
A07F.FFFFH
8 MBYTE
LOCAL TO PCI APERTURE 0
A000.0000H
0000.0000H
ADDRESS PATH
LOCAL TO PCI APERTURE 1 ADDRESS REMAPPER AND PCI COMMAND INSERTION
DATA PATH
LOCAL TO PCI APERTURE 1 BYTE ORDER CONVERTER
READ ADDRESS
WRITE ADDRESS
WRITE DATA
READ DATA
APERTURE 0 AND 1 LOCAL TO PCI WRITE FIFO (256 BYTES)
APERTURE 1 LOCAL TO PCI READ FIFO (16 BYTES)
PCI ADDRESS GENERATOR
PCI DATA
TO PCI BUS CONTROLLER
APERTURE 0 LOGIC OMMITTED FOR CLARITY.
APERTURES 0 AND 1 SHARE THE LOCAL-TO-PCI WRITE FIFO.

Figure 4: Block Diagram of Local-to-PCI Aperture/FIFO Connections

Figure 5: Block Diagram of a PCI-to-Local Aperture/FIFO Connections


PCI SPACE
FFFF.FFFFH
E07F.FFFFH
8 MBYTE
E000.0000H
0000.0000H
PCI TO LOCAL APERTURE 1
PCI TO LOCAL APERTURE 0
ADDRESS PATH
PCI TO LOCAL APERTURE 1 ADDRESS REMAPPER
DATA PATH
PCI TO LOCAL APERTURE 1 BYTE ORDER CONVERTER
READ ADDRESS
WRITE ADDRESS
WRITE DATA
READ DATA
APERTURE 0 AND 1 PCI TO LOCAL WRITE FIFO (256 BYTES)
APERTURE 1 PCI TO LOCAL READ FIFO (16 BYTES)
LOCAL ADDRESS GENERATOR
LOCAL DATA
TO LOCAL BUS CONTROLLER
APERTURE 0 LOGIC OMMITTED FOR CLARITY.
APERTURES 0 AND 1 SHARE THE PCI-TO-LOCAL WRITE FIFO.

4.1.2 Selecting PCI Memory or I/O Space Mapping

The PCI-to-Local apertures may be mapped into PCI memory space or PCI I/O space. The IO bit in the PCI_BASEx registers controls this mapping. When IO=1, the aperture will only "capture" transfers on the PCI bus that are to I/O space and fall within the bounds set by the address base and size. Similarly, when IO=0, the corresponding PCI-to-Local aperture will only respond to memory space transfers on the PCI bus.

4.1.3 PCI-to-Local Address Translation

PCI-to-Local address translation is controlled via the MAP_ADDR field in the PCI_MAPx register. When an access is bridged from PCI-to-Local, the upper address bits of the PCI address are *always* replaced with the significant bits in the MAP_ADDR field. Which bits are considered "significant" is controlled by the size of the aperture. For example, with a 1 megabyte aperture size, the EPC will create the local bus address by replacing the A[31:20] of the PCI address with the entire MAP_ADR field. With larger apertures, less of upper address bits are replaced. For example with a 2 megabyte aperture, only PCI address bits A[31:21] will be replaced. Address remapping is "disabled" by simply setting the MAP_ADR and ADR_BASE fields for a particular aperture to the same value.

Table 2: PCI Address Remapping By Aperture Size[a]

Aperture Size	PCI Address Bits Replaced by MAP_ADR Bits
1 meg	A[31:20]
2 meg	A[31:21]
4 meg	A[31:22]
8 meg	A[31:23]
16 meg	A[31:24]
32 meg	A[31:25]
64 meg	A[31:26]
128 meg	A[31:27]
256 meg	A[31:28]

[a] Some combinations are used for special DOS Compatibility apertures.

4.1.4 Byte Order Conversion

The PCI-to-Local bus apertures also control the conversion of data byte ordering as data flows across the bridge. Writes from PCI space to local space are converted before entry into the write FIFO, reads from local space destined for PCI space are converted on their way from FIFO to the PCI bus (see Figure 5). Three modes of endian conversion are supported as is shown in Table 3. When byte order conversion is enabled, the byte enable signals are automatically corrected for the bridge transfer.

Byte order conversion is controlled through the SWAP field in the PCI_MAPx registers.

Table 3: Byte Order Conversion Options

	Input Data Bytes				
Swap Mode	**D[31:24]**	**D[23:16]**	**D[15:8]**	**D[7:0]**	
32-bit (no swap)	D[31:24]	D[23:16]	D[15:8]	D[7:0]	Output Data Bytes
16-bit (half-word swap)	D[15:8]	D[7:0]	D[31:24]	D[23:16]	
8-bit (word swap)	D[7:0]	D[15:8]	D[23:16]	D[31:24]	

4.1.5 Enabling Read Prefetching

The EPC is capable of prefetching data for PCI-to-Local bus reads. Prefetching will often improve performance in applications that perform many sequential reads from the same aperture. Prefetching is discussed in more detail in “FIFO Architecture”.

Read prefetching is enabled for a PCI-to-Local aperture by setting the PREFETCH bit in the PCI_BASEx register.

4.1.6 Disabling PCI-to-Local Bus Apertures

The PCI specification does not provide a method to explicitly disable PCI apertures that are implemented as *base registers*. Disabling of the PCI-to-Local apertures is achieved by programming the ENABLE bit in the PCI_MAPx registers.

4.1.7 Overlapping Apertures

If data transfer apertures 0 and 1 overlap, the EPC will use aperture 0 (aperture 0 has priority). PCI_IO_BASE has the lowest priority if it overlaps either of the PCI_BASEx regions.

4.1.8 Special Function Modes for PCI-to-Local Bus Apertures

PCI-to-Local bus aperture 0 shares some functionality with the expansion ROM base aperture (see “PC Compatibility”). The address decoder for PCI-to-Local aperture 0 is shared with the expansion ROM base register. When the expansion ROM base is enabled, the decoder will only bridge accesses within the ROM window. When the ROM is disabled, PCI-to-Local bus aperture 0 will function as described above. Typically, the expansion ROM is used only during BIOS boot, if at all. The expansion ROM base register can be completely disabled via software.

PCI-to-Local bus aperture 1 includes special logic for compatibility with legacy DOS systems. These functions are described in the “PC Compatibility” chapter.

4.2 LOCAL-TO-PCI BUS APERTURES

The Local-to-PCI bus apertures control the following accesses: writes from local memory to PCI space and reads from PCI space destined for the local processor/memory space.

The programming of the Local-to-PCI apertures is controlled via the LB_BASEx and LB_MAPx registers. Many of the features for the Local-to-PCI apertures are identical to those for the PCI-to-Local apertures. The following options are programmable for each Local-to-PCI aperture:

- Base address of aperture
- Aperture size
- Type of PCI command generated (memory, I/O, configuration, etc.)
- Address translation
- Endian conversion
- Read prefetch enable/disable
- Local-to-PCI aperture enable/disable

4.2.1 Setting the PCI Command Type

Each PCI address cycle includes information about the type of access being attempted (i.e. memory, I/O, configuration). This "type of access" information is also called a PCI command, and is encoded on the C/$\overline{\text{BE}}$[3:0] lines of the PCI bus.

On the local side of the bridge, the EPC sees only memory reads and writes. The PCI bus, however, supports many types of accesses. When an accesses is "captured" on the local bus it must be converted into a specific type of PCI access. This conversion is controlled by the TYPE bits in the LB_MAPx register. During the address phase of a PCI access, the TYPE bits are copied directly to the C/$\overline{\text{BE}}$[3:1] pins, while C/$\overline{\text{BE0}}$ is set according to the direction of the access (read or write). Table 4 shows the command encodings per the revision 2.1 PCI specification and the appropriate TYPE field entries to generate these encodings.

The EPC will generate commands that are "reserved" in the specification, and performs no error checking on the validity of the encodings. The EPC will also generate a "Dual Address Cycle (DAC)" command if programmed to do so, however, Dual Addressing is not supported by the bridge and thus will fail during the second phase of a dual address cycle.[1]

Table 4: PCI Command Encodings and Corresponding TYPE Field Values

C/$\overline{\text{BE[3:0]}}$	Command	TYPE Field Value
0000	Interrupt Acknowledge	000
0001	Special Cycle	000
0010	I/O Read	001
0011	I/O Write	001
0100	reserved	010
0101	reserved	010
0110	Memory Read	011
0111	Memory Write	011
1000	reserved	100
1001	reserved	100
1010	Configuration Read	101
1011	Configuration Write	101
1100	Memory Read Multiple	110
1101	Dual-Address Cycle (DO NOT USE)	110
1110	Memory Read Line	111
1111	Memory Write and Invalidate	111

4.2.2 Setting the Local-to-PCI Aperture Base Address and Size

The base address for a Local-to-PCI memory aperture is set in the ADR_BASE field of the LB_BASEx register. Only AD[31:20] are significant yielding a minimum base address granularity of 1Mbyte.

The size of a Local-to-PCI aperture is set via the ADR_SIZE field in the LB_BASEx register. Supported sizes are from 1 to 256[1] megabytes increasing as powers-of-2 (1M, 2M, 4M, etc.).

[1] DAC is used to support 64-bit targets on a 32-bit PCI bus. It requires that the 64-bit address be given in two back-to-back address phases followed by the burst 32-bit data phases. See the PCI specification for more details. THE EPC DOES NOT SUPPORT DAC.

[1] Future versions of the EPC will allow the Local-to-PCI apertures size up to 2Gb (on 512Mb boundary), and the PCI-to-Local aperture size up to 1Gb (on 1Gb boundary).

4.2.3 Local-to-PCI Address Translation

Local-to-PCI address translation is controlled via the MAP_ADDR field in the LB_BASEx register. Address remapping from local-to-PCI is implemented identically to remapping in the PCI-to-Local direction. Please see “PCI-to-Local Address Translation” on page 21 for details.

4.2.4 Byte Order Conversion

Byte order conversion for local-to-PCI transfers is controlled through the SWAP field in the LB_BASEx registers. For a description of byte order conversion, please see the PCI-to-Local description above.

4.2.5 Enabling Read Prefetching

The EPC is capable of prefetching data for Local-to-PCI bus reads. Prefetching will often improve performance in applications that perform many sequential reads from the same aperture. Prefetching is discussed in more detail in the "FIFO Operation" section of this chapter.

Read prefetching is enabled for a Local-to-PCI aperture by setting the PREFETCH bit in the LB_BASEx register.

4.2.6 Enabling Local-to-PCI Bus Apertures

Unlike the PCI-to-Local registers, it is possible to explicitly enable and disable Local-to-PCI apertures. The Local-to-PCI apertures are **disabled following a reset** and must be enabled before the EPC will recognize Local-to-PCI transfers. To enable an Local-to-PCI aperture, set the ENABLE bit in the appropriate LB_BASEx register.

Chapter 5 FIFO Architecture and Operation

The Local bus and PCI bus are decoupled from each other through the use of FIFOs. The FIFOs provide “elastic” storage for transfers passing across the bridge. The FIFOs also provide the synchronization necessary when running the PCI bus and the Local bus at different frequencies.

FIFOs are necessary to prevent performance bottlenecks that would arise if the Local and PCI buses were connected directly. As an example, imagine a bridge with no FIFO storage. When a local master wanted to write to the PCI bus, it would have to wait (i.e. be held NOT READY) until the PCI bus was available before continuing operation. With FIFOs however, the local master simply writes data into the FIFO and expects the bridge to complete the transfer at a later time (*write posting*).

The size of the FIFO storage in a PCI bridge is very important, especially in high-bandwidth applications. It is very possible that a small FIFO could be filled by a single data transfer, leaving one of the buses "hanging" and seriously degrading system performance. The EPC bridge includes a large amount of FIFO storage to prevent such situations from occurring.

Another important capability provided by the EPC FIFOs is called *Dynamic Bandwidth Allocation*. This feature allows the programmer to control precisely how empty, or full the FIFOs get before initiating a data transfer as well as setting the priority between reads and writes in each direction. Dynamic Bandwidth Allocation is critical for applications using high-bandwidth peripherals such as advanced networking, communications, and graphics devices. The EPC also includes FIFO performance monitoring logic to allow the tuning of system code to maximize performance.

5.1 DYNAMIC BANDWIDTH ALLOCATION FIFO ARCHITECTURE

The FIFO architecture is logically divided in two blocks: one for PCI-to-Local transfers, another for Local-to-PCI transfers. The FIFO architecture is symmetrical: the same amount of buffering and programmability are provided in each direction. The FIFO organization is shown in the block diagram in Figure 6.

There are three FIFOs within the Local-to-PCI FIFO block:

- **Local-to-PCI Write FIFO (256 bytes)**. Buffers local bus writes to local bus data transfer apertures 0 and 1. Also buffers DMA writes to the PCI bus. This FIFO contains the initial address and data for a transaction.

- **PCI Read Aperture 0 FIFO (32 bytes)**. Buffers PCI bus reads from local memory issued through PCI aperture 0. This FIFO also buffers prefetch reads from local memory when this feature is enabled.

- **PCI Read Aperture 1 FIFO (32 bytes)**. Buffers PCI bus reads from local memory issued through PCI aperture 1. This FIFO also buffers fetch-ahead reads from local memory when this feature is enabled.

In addition, there are three FIFOs within the PCI-to-Local FIFO block:

- **PCI-to-Local Write FIFO (256 bytes)**. Buffers PCI bus writes to PCI bus apertures 0 and 1. Also buffers DMA writes to the local bus. This FIFO contains both the address and data for a transaction.

- **Local Read Aperture 0 FIFO (32 bytes)**. Buffers local bus reads from PCI space issued through local bus aperture 0. This FIFO also buffers fetch-ahead reads from PCI space when this feature is enabled.

- **Local Read Aperture 1 FIFO (32 bytes)**. Buffers local bus reads from PCI space issued through PCI aperture 1. This FIFO also buffers fetch-ahead reads from PCI space when this feature is enabled.

The “Dynamic Bandwidth Allocation” feature is the ability to use the large 256-byte FIFO in each direction for multiple transfers; both DMA and Aperture. At any one instant in time, a FIFO can contain 1 or more of the following items:

- Aperture 0 posted writes

- Aperture 1 posted writes

- DMA 0 data

- DMA 1 data

5.2 WRITE FIFO OPERATION AND PROGRAMMING

Each write FIFO stores address, byte enable, and data information for write transfers bridged across the EPC. The operation and programming of the PCI-to-Local and Local-to-PCI write FIFOs are nearly identical. For brevity, this section will refer to a generic “write FIFO operation”.

The write FIFOs are used for the following transactions:

- Writes to either data transfer aperture 0 or aperture 1

- DMA transfers destined for the PCI (local) bus

When an access is bridged from one bus to the other, the starting address of the access is captured, translated, and then stored in the write FIFO. Subsequent data cycles are also captured, byte-order converted, and then stored in the write FIFO as well. The state of the byte enable lines is also captured, byte-order corrected and then stored with the data.[1] This process is shown in Figure 7 for a four word burst write from Local-to-PCI with a starting address of 1000.4020H.

For writes from Local-to-PCI the command type to be generated for the PCI write is also generated and stored in the Local-to-PCI write FIFO. For example, if you have programmed Local-to-PCI aperture 1 to generate "Configuration" commands, then as the address for Local-to-PCI Aperture 1 writes passes through the Local-to-PCI Aperture 1 address remapper it will be "tagged" with the type of PCI command to be generated. This information is then stored in the Local-to-PCI FIFO.

5.2.1 Write FIFO Draining Strategies

The target bus for a write transaction must be requested by the EPC in order for the write to complete. The $\overline{\text{REQ}}$/$\overline{\text{GNT}}$ protocol is used on the PCI bus for this purpose. The local bus mastership protocol is dependent on the processor selected and is described in greater detail in the "Local Bus Interface" chapter. How rapidly the target bus is requested is programmable through the selection of a write FIFO *draining strategy.* The options for draining strategies are shown in Table 5.

The choice of strategy is highly application dependent. Draining strategy "00" will result in the fastest transfer of data from one bus to the other; however, it will also result in the most bus traffic. Strategy "10" has the advantage of preserving bus bandwidth, but will allow small transfers (< 3 words) to sit in the write FIFO indefinitely. Strategy "11" will preserve bus bandwidth, but will complete the write by requesting the bus as soon as the data write transaction filling the FIFO is complete (i.e. data will not sit in the FIFO indefinitely). The write FIFO draining strategies are controlled via the FIFO_CFG register.

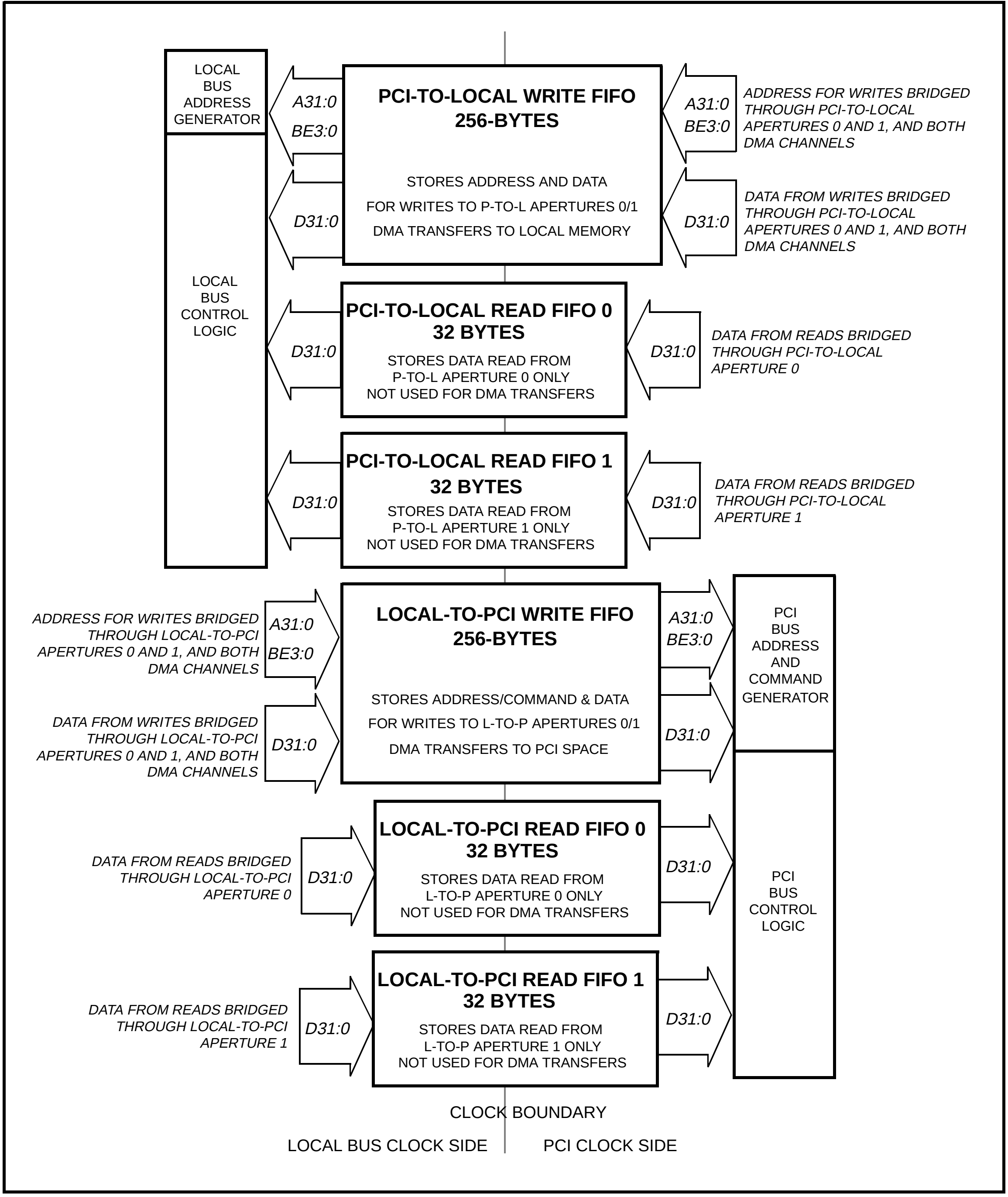
.

Table 5: Summary of Write FIFO Draining Strategies

Strategy	Description
00	Request target bus whenever corresponding write FIFO is not empty
01	Reserved: do not use
10	Request target bus when 3 or more words of data have been posted in the corresponding write FIFO
11	Request target bus when 3 or more words of data have been posted in the write FIFO *or* a burst write has been completed at the "filling end" of the FIFO

[1] The storage space for the byte enable state is in addition to the 256-byte FIFO size.

Figure 6: FIFO Architecture


LOCAL BUS ADDRESS GENERATOR
LOCAL BUS CONTROL LOGIC
PCI-TO-LOCAL WRITE FIFO 256-BYTES
STORES ADDRESS AND DATA FOR WRITES TO P-TO-L APERTURES 0/1 DMA TRANSFERS TO LOCAL MEMORY
A31:0 BE3:0
D31:0
ADDRESS FOR WRITES BRIDGED THROUGH PCI-TO-LOCAL APERTURES 0 AND 1, AND BOTH DMA CHANNELS
DATA FROM WRITES BRIDGED THROUGH PCI-TO-LOCAL APERTURES 0 AND 1, AND BOTH DMA CHANNELS
PCI-TO-LOCAL READ FIFO 0 32 BYTES
STORES DATA READ FROM P-TO-L APERTURE 0 ONLY NOT USED FOR DMA TRANSFERS
DATA FROM READS BRIDGED THROUGH PCI-TO-LOCAL APERTURE 0
PCI-TO-LOCAL READ FIFO 1 32 BYTES
STORES DATA READ FROM P-TO-L APERTURE 1 ONLY NOT USED FOR DMA TRANSFERS
DATA FROM READS BRIDGED THROUGH PCI-TO-LOCAL APERTURE 1
LOCAL-TO-PCI WRITE FIFO 256-BYTES
STORES ADDRESS/COMMAND & DATA FOR WRITES TO L-TO-P APERTURES 0/1 DMA TRANSFERS TO PCI SPACE
ADDRESS FOR WRITES BRIDGED THROUGH LOCAL-TO-PCI APERTURES 0 AND 1, AND BOTH DMA CHANNELS
DATA FROM WRITES BRIDGED THROUGH LOCAL-TO-PCI APERTURES 0 AND 1, AND BOTH DMA CHANNELS
PCI BUS ADDRESS AND COMMAND GENERATOR
PCI BUS CONTROL LOGIC
LOCAL-TO-PCI READ FIFO 0 32 BYTES
STORES DATA READ FROM L-TO-P APERTURE 0 ONLY NOT USED FOR DMA TRANSFERS
DATA FROM READS BRIDGED THROUGH LOCAL-TO-PCI APERTURE 0
LOCAL-TO-PCI READ FIFO 1 32 BYTES
STORES DATA READ FROM L-TO-P APERTURE 1 ONLY NOT USED FOR DMA TRANSFERS
DATA FROM READS BRIDGED THROUGH LOCAL-TO-PCI APERTURE 1
CLOCK BOUNDARY
LOCAL BUS CLOCK SIDE
PCI CLOCK SIDE

5.3 READ FIFO OPERATION AND PROGRAMMING

Two 16-byte read FIFOs are provided for transfers in each direction. Each transfer aperture has a separate read FIFO as shown in Figure 6. Like the write FIFOs, the operation of each read FIFO is identical.

The read FIFOs store data for the following transactions:

- PCI reads from local space through PCI-to-Local apertures 0 and 1
- Local bus reads from PCI space through Local-to-PCI apertures 0 and 1

The read FIFOs serve two purposes: to allow synchronization between the PCI and Local buses and to provide storage for read prefetching. The read FIFOs only store the data for read transfers, the address is not stored since reads request the target bus immediately (i.e. reads are not “posted”).

5.3.1 Prefetching and Read FIFO Filling Strategies

Performance for reads from sequential locations can be improved by enabling read prefetching. With prefetching enabled, a read FIFO will “guess” that additional read transfers will occur and will perform burst reads to fill the FIFO with data from subsequent locations. The aggressiveness of the prefetch is controlled via a programmable “filling strategy” for each read FIFO. The filling strategies are controlled via the FIFO_CFG register.

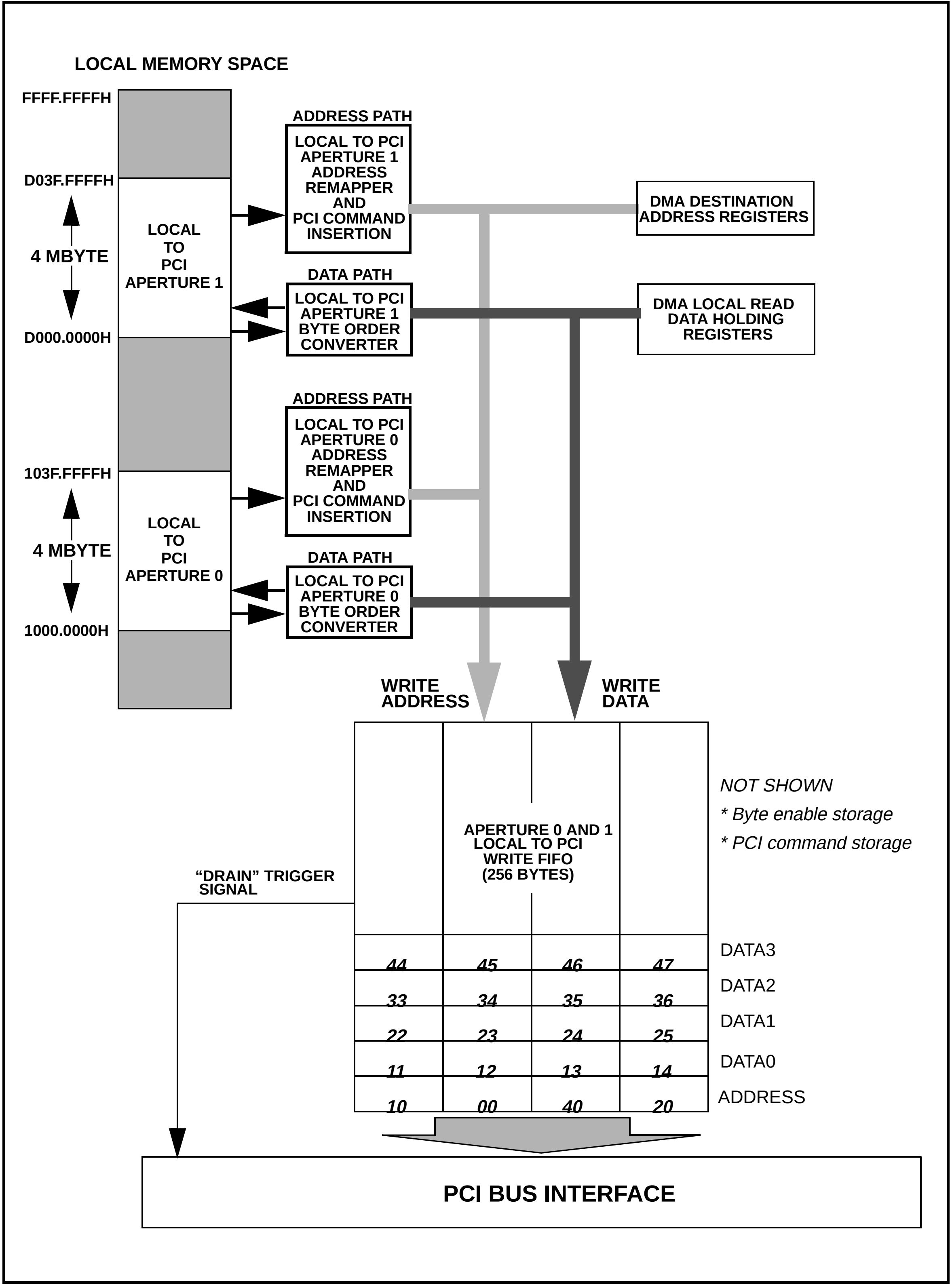
As an example, let’s assume prefetching is enabled for Local-to-PCI aperture 0 (active from 1000.0000H to 107F.FFFFH, no address remapping). When the local bus master performs a read from location 1000.0020H, the Local-to-PCI FIFO 0 will initiate a PCI burst read from 1000.0020H to 1000.002FH (4 words). The read FIFO will continue to fill itself as long as its programmable “filling strategy” dictates.

The read FIFOs contain an address comparator that is used to determine whether or not a new read request can be serviced from data already fetched. For example, if the Local CPU performs a two word burst access from 100H and 104H, the read FIFO will continue to fetch ahead from locations 108H 10CH, etc. If the Local CPU then performs a burst read from 108H and 10CH, the EPC will supply the data directly from the read FIFO. If the next access to a read FIFO is not the next highest word address, then all entries in the read FIFO are invalidated and the EPC fetches new data from the PCI (or Local) bus.

Table 6: Summary of Read FIFO Filling Strategies

Strategy	Description
00	Request the PCI (or Local) bus whenever there is room for at least 1 word in the read FIFO
01	Request the PCI (or Local) bus whenever there is room for at least 2 words in the read FIFO
10	Request the PCI (or Local) bus whenever the read FIFO is empty
11	Reserved: do not use

Figure 7: Detailed Operation of the Local-to-PCI Write FIFOs



5.4 FIFO PRIORITIZATION OPTIONS

The EPC allows reads and writes bound for completion on the same bus to be prioritized with respect to each other. This capability is critical in systems with many PCI or local bus masters.

For example, let's assume that a number of writes have been posted in the Local-to-PCI FIFO while waiting for the EPC to be granted the PCI bus. Subsequently, the local bus master requests a read from the PCI bus. When the EPC is finally granted the PCI bus, which transfer proceeds? If the EPC were designed like many bridges, the answer would be "first in first out"; in other words the local processor would be forced to wait while all of the writes in the Local-to-PCI write FIFO had completed. Luckily, the EPC includes prioritization options that allow the read to proceed first, "unlocking" the local bus and allowing the local CPU to get back to work.

FIFO prioritization options are described in Table 7, below.

Table 7: Read/Write Prioritization Options

Prioritization Option	Result
Local bus reads ahead of writes	Pending local bus reads will complete before writes are allowed to complete. Prevents PCI bus from experiencing multiple disconnect/retries while attempting a read of local memory. May result in data coherency hazards (see below).
Local bus writes ahead of reads	Pending local bus writes will complete ahead of reads. Prevents data coherency hazards, but may cause PCI bandwidth degradations.
PCI bus reads ahead of writes	Pending PCI bus reads will complete before writes are allowed to complete. Prevents local bus from experiencing extended lockup while attempting a read of PCI space. May result in data coherency hazards (see below).
PCI bus writes ahead of reads	Pending PCI bus writes will complete ahead of reads. Prevents data coherency hazards, but may cause local bus bandwidth and CPU performance degradation.

The FIFO priority is only considered by the EPC on arbitration boundary. For example, when reads and writes are pending, the priority decision will be made after EPC's HOLDA is deasserted.

5.5 FIFO DATA COHERENCY OPTIONS

Some applications will require strict data coherency. With programmable FIFOs priority, it is possible that a write to location in memory may be prioritized behind a read to the same location that occurred later in time. Allowing such a read to proceed would result in "stale" data being returned to the initiator of the read. The EPC provides a programmable data coherency mechanism to prevent this situation from occurring. Table 8 shows the coherency mechanisms available. Coherency options are set through the FIFO_PRIORITY register.

In addition to the options shown in Table 8, all FIFOs can be flushed immediately under program control. Direct FIFO data flushing is performed by writes to appropriate “flush” bits in the SYSTEM register. A FIFO flush results in all data within the FIFO being invalidated. This is true of the write FIFOs as well: flushing a write FIFO *does not* result in the data in the FIFO being written to the target bus (the data in the FIFO is simply “thrown away”). Using this method to flush does NOT alter the state machines that fill or drain the FIFO. Consequently, this method of flushing should not be used under normal operations. This flushing mechanism is intended to be used only for test or fault recovery situations.

When prefetching is disabled for a particular aperture, then data flushing for that aperture should be disabled. This does **not** create a coherency problem because when prefetching disabled stale data can never remain sitting in the FIFO (only the exact amount of data asked for will be fetched). Enabling flush for a non-prefetch aperture will result in unpredictable read behaviour. .

Table 8: FIFO Data Coherency Options

Coherency Strategy	Programmable Options for Flushing the READ FIFO
Local Bus Aperture 1 Read-Ahead FIFO Flush Strategy	· Local bus to PCI writes never cause a flush · Local bus to PCI writes to aperture 1 *only* cause a flush · Local bus to PCI writes to either aperture cause a flush
Local Bus Aperture 0 Read-Ahead FIFO Flush Strategy	· Local bus to PCI writes never cause a flush · Local bus to PCI writes to aperture 0 *only* cause a flush · Local bus to PCI writes to either aperture cause a flush
PCI Bus Aperture 1 Read-Ahead FIFO Flush Strategy	· PCI to local bus writes never cause a flush · PCI to local bus writes to aperture 1 *only* cause a flush · PCI to local bus writes to either aperture cause a flush
PCI Bus Aperture 0 Read-Ahead FIFO Flush Strategy	· PCI to local bus writes never cause a flush · PCI to local bus writes to aperture 0 *only* cause a flush · PCI to local bus writes to either aperture cause a flush

5.5.1 Ensuring Strict Data Coherency

In systems where data coherency must be strictly maintained, the following options should be selected:

- If prefetch is enabled, cause a flush of the read prefetch FIFO(s) whenever a write occurs by programming FIFO_PRIORITY for LB_RD0,1/PCI_RD0,1 = “11”

- Program FIFO_PRIORITY so that writes have priority over reads

- Program FIFO_CFG so that the write FIFO drain strategy is “00” or “11” (this will prevent write data from sitting in the FIFO)

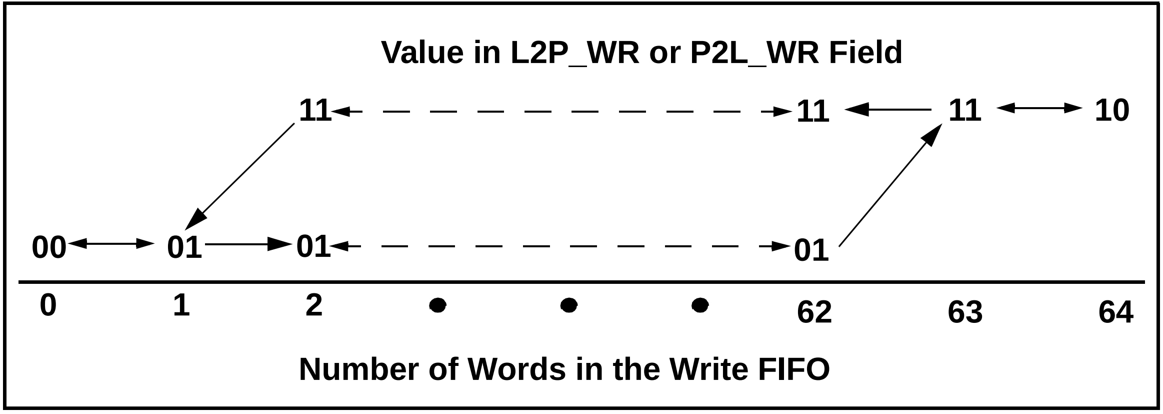
5.5.2 Monitoring the Status of Read and Write FIFOs

The two write FIFOs and four read FIFOs each provide an indication of their relative "fullness" through bits in the FIFO_STATUS register. This information can be used to help tune "filling" and "draining" strategies, as well as to determine when there is room available in the FIFOs for additional transactions.

The status bits for the write FIFOs indicate both the fullness of the FIFOs as well as whether the FIFOs are in the process of filling or draining. Two bits are used in the P2L_WR and L2P_WR fields in the FIFO_STATUS register. A description of these bits is shown in Figure 8.

Each write FIFO has a two bit flag that indicates whether there is room for additional data. The read FIFO status bit encoding is shown in the *Register Descriptions* chapter.

Figure 8: Write FIFO Status Bits


Value in L2P_WR or P2L_WR Field
11
11
11
10
00
01
01
01
0
1
2
●
●
●
62
63
64
Number of Words in the Write FIFO

5.5.3 Ensuring the Completion of a Posted Write

Occasionally, the system software will need to ensure that a posted write has completed on the PCI bus. There are two methods available:

- **Software Polling.** Before issuing the write transfer, wait for the corresponding write FIFO to become empty (by monitoring the FIFO status bits). Then post the write and begin monitoring the status bits again. It is a good idea to check the PCI_STATUS register to see if any errors occurred (e.g. a Master or Target Abort).

- **Hardware Stall**. Program the FIFO priority for writes-ahead-of-reads. First post the write in the write FIFO, then attempt a dummy read from the same aperture. The dummy read will lock the local bus until the write completes. (Note: systems using V3 Semiconductor memory controllers must take into account the effect of the bus watch timers if those features are enabled!).

5.6 FIFO LATENCY

FIFO latency is defined as the amount of time necessary for a word of data to flow through a given FIFO from source to destination. Since the EPC supports fully asynchronous operation of the PCI and local interfaces, exact latency is impossible to specify and is not deterministic.

The only way to specify best case latencies is to assume both interfaces run at the same frequency with little or no skew between clocks. In such a situation the latency for data through the bridge will range from 2-4 clock cycles. An exact time cannot be given since there is no internal synchronization between the PCI and local bus clocks within the EPC.

Chapter 6 DMA Controller

The EPC's DMA Controller includes two channels, each capable of transferring data from Local memory to PCI, or from PCI to Local memory. The DMA Controller supports the following features:

- Block transfers up to 4 megabytes in size for each link
- On-the-fly byte order conversion
- PCI and local transfer ranges independent of data transfer apertures
- Block chaining with no limit on the number of chained blocks

The DMA controller is useful in applications that transfer large amounts of sequential data across the bridge. For example, an intelligent disk controller may need to transfer a data buffer from the local memory space to the host processors' memory space located on the PCI bus. Using DMA is ideal in this case, since the programmer simply sets starting addresses and the transfer count, then lets the EPC transfer the data without local or host processor intervention.

6.1 DMA TRANSFERS

A DMA transfer consists of the movement of data by the EPC from local memory to the PCI bus, or from PCI memory to the local bus.

6.1.1 Local Bus to PCI Bus DMA Transfers

Prior to starting local-to-PCI DMA transfers, the programmer must set the local and PCI starting address, as well as the byte order conversion, direction, and priority options. DMA transfers are initiated by setting the IPR bit within the DMA_CSRx register.

A Local-to-PCI DMA data transfer begins with the EPC requesting the local bus. Once the local bus is granted, the EPC performs a local bus read to fetch the contents of the memory location pointed to by the local DMA address register (DMA_LOCAL_ADDRx). The local bus read performed by the DMA controller does not use the read prefetch FIFOs.The data read from the local bus is then placed in the Local-to-PCI write FIFO, the address for the write phase is stored in the DMA address generator. The PCI command that is used for the write cycle can be programmed via the DMA_WTYPE bits (DMA Write to PCI Bus Command Type) in the PCI_CFG register. It defaults to a value of 3h which produces a memory write command of "0111".

After each local bus read transfer has completed, the local address is incremented and the transfer count register is decremented. The DMA controller will continue to repeat the above process until the transfer count reaches zero. Since transfer count is tracked on the source *read* cycle (as opposed to the destination *write* cycle), polling the transfer count register for a zero value is NOT a reliable way to ensure that the transfer is complete since the data may not yet be written to the destination and may be sitting in the FIFO. To determine that a DMA transfer is finished and it is safe to reprogram it, the DMA_IPR bit in the DMA_CSRx register should be polled for ’0’. If DMA_IPR is not clear, the contents of the destination address should not be changed.

When the transfer count has reached zero, the DMA controller will either generate a “process complete” interrupt, or fetch the next block transfer descriptor (if programmed for chaining). Block chaining is described below.

6.1.2 PCI Bus to Local Bus DMA Transfers

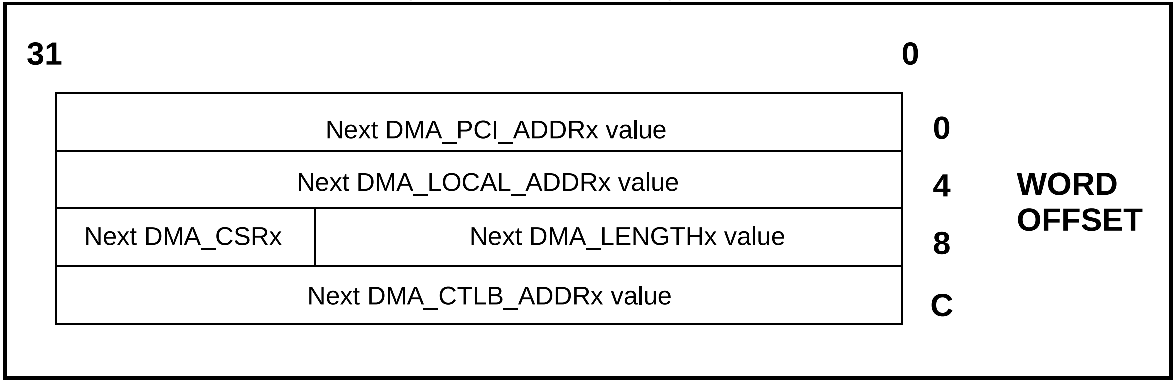
PCI to Local bus DMA transfers operate nearly identically to the Local to PCI DMA transfers described above. In this case, the data is read from the PCI side of the bridge (using a PCI “Read Memory” command) and then posted in the PCI-to-Local write FIFO.

A flowchart of basic DMA operation is shown in Figure 10.

6.1.3 DMA Block Chaining

Upon completion of a single block transfer, the DMA Controller can be programmed to automatically fetch a new control block descriptor with the parameters for a subsequent block. The descriptor includes the values for the PCI start address, Local start address, transfer length, next control block descriptor address and control information. The initial control block address is stored in the DMA_CTLB_ADRx register for each channel. DMA control blocks may be located in local memory only. Figure 9 shows the layout of the DMA chaining descriptor. Please note that the format for the chaining descriptor is for little-endian memory, as this is the same format used within the EPC’s control registers.

Figure 9: DMA Chaining Descriptor Layout.



31	0	WORD OFFSET
Next DMA_PCI_ADDRx value		0
Next DMA_LOCAL_ADDRx value		4
Next DMA_CSRx	Next DMA_LENGTHx value	8
Next DMA_CTLB_ADDRx value		C

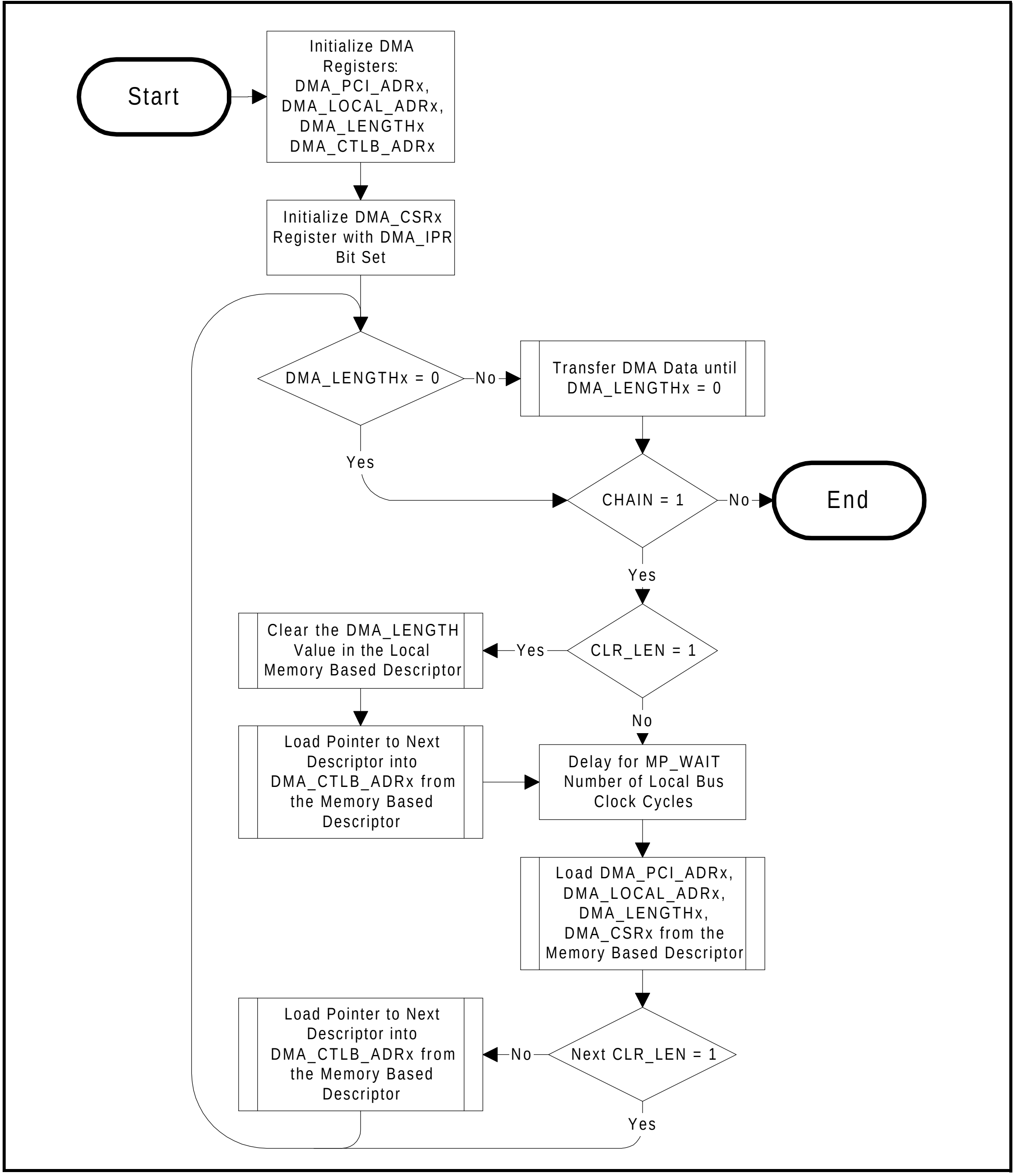
DMA chaining is controlled by the CHAIN bit in the DMA_CSRx register. Block chaining transfers begin the same as normal DMA transfers: the first block's start addresses, transfer count, and control parameters are set up by the programmer in the EPC's DMA registers. When the first transfer completes, the DMA controller checks the CHAIN bit, and if this bit is set, it will fetch the descriptor pointed to by the DMA_CTLB_ADRx register. Chaining terminates when the DMA controller completes a block for which the CHAIN bit is not set. There is no limit on how many blocks may be chained together.

6.1.4 Multi-processor DMA Chaining

A special bit in the DMA_CSR registers (called CLR_LEN) can be used to cause the length value of the memory based DMA descriptor to be cleared once that descriptor has been processed by the DMA controller. Therefore, looped DMA chains may be created without redundant transfers taking place. In a multi-processor environment, this can be used to create a large number of virtual DMA channels. A particular processor or process can be assigned one or more of these virtual channels. They can then initiate a transfer by setting up the memory based descriptor as desired and then updating the DMA_LENGTH portion of the descriptor last. The next time that descriptor is processed then DMA will be performed and then the DMA_LENGTH value cleared. Then when the looped descriptors are processed on the next pass, no data transfer will be initiated.

In order to avoid having the DMA engine polling to aggressively on the local memory when the descriptors are looped, the DMA_DELAY register is provided. This register controls the number of clocks of delay between when one descriptor finishes and another is loaded. A larger value will provide more time for other local bus masters to occupy the local bus.

Figure 10: Flowchart of Basic DMA Operation


Start
Initialize DMA Registers: DMA_PCI_ADRx, DMA_LOCAL_ADRx, DMA_LENGTHx DMA_CTLB_ADRx
Initialize DMA_CSRx Register with DMA_IPR Bit Set
DMA_LENGTHx = 0
No
Transfer DMA Data until DMA_LENGTHx = 0
Yes
CHAIN = 1
No
End
Yes
CLR_LEN = 1
Yes
Clear the DMA_LENGTH Value in the Local Memory Based Descriptor
No
Load Pointer to Next Descriptor into DMA_CTLB_ADRx from the Memory Based Descriptor
Delay for MP_WAIT Number of Local Bus Clock Cycles
Load DMA_PCI_ADRx, DMA_LOCAL_ADRx, DMA_LENGTHx, DMA_CSRx from the Memory Based Descriptor
Next CLR_LEN = 1
No
Load Pointer to Next Descriptor into DMA_CTLB_ADRx from the Memory Based Descriptor
Yes

6.1.5 DMA Transfer Size

The DMA controller performs all transfers in word (32 bits) sizes.

For the V350EPC where the local bus is 16-bit wide, a 16 to 32-bit conversion will occur (the EPC will wait for a second 16-bit word before writing a 32-bit word to a PCI bus). DMA transfers begin and end on a 32-bit boundary and local bus transfers will be done 16 bits at a time (one data phase on the PCI bus will produce two data phases on the local bus). Byte and short (16-bit) boundaries are not supported. There is no performance penalty for this restriction. From a software standpoint, a programmer wishing to transfer byte data need only normalize the byte pointers to the next inclusive word boundary. This will result in transferring more *byte* data than necessary, however, since the PCI bus transfers data in 32-bit words, there is no negative performance effect.

6.1.6 Relationship to the Data Transfer Apertures

The DMA Controller does not use the data transfer apertures. The write FIFOs, however, are shared between transfers initiated by a bus master through either the PCI-to-Local or Local-to-PCI apertures, and by the DMA Controller. Figure 11 shows the usage of the write FIFOs by both the DMA Controller and the data transfer apertures.

The read ahead FIFOs are *only* used during bus master read accesses through the data transfer apertures; they are not used during DMA operations at all.

6.1.7 Automatic DMA Throttling

The DMA Controller has a built in throttling mechanism to prevent it from monopolizing the write FIFO or target buses. The DMA Controller will not initiate a transfer if the target write FIFO is more than half full. Once the write FIFO is drained below one quarter full, DMA transfers involving the corresponding write FIFO will proceed.

6.1.8 Demand Mode DMA

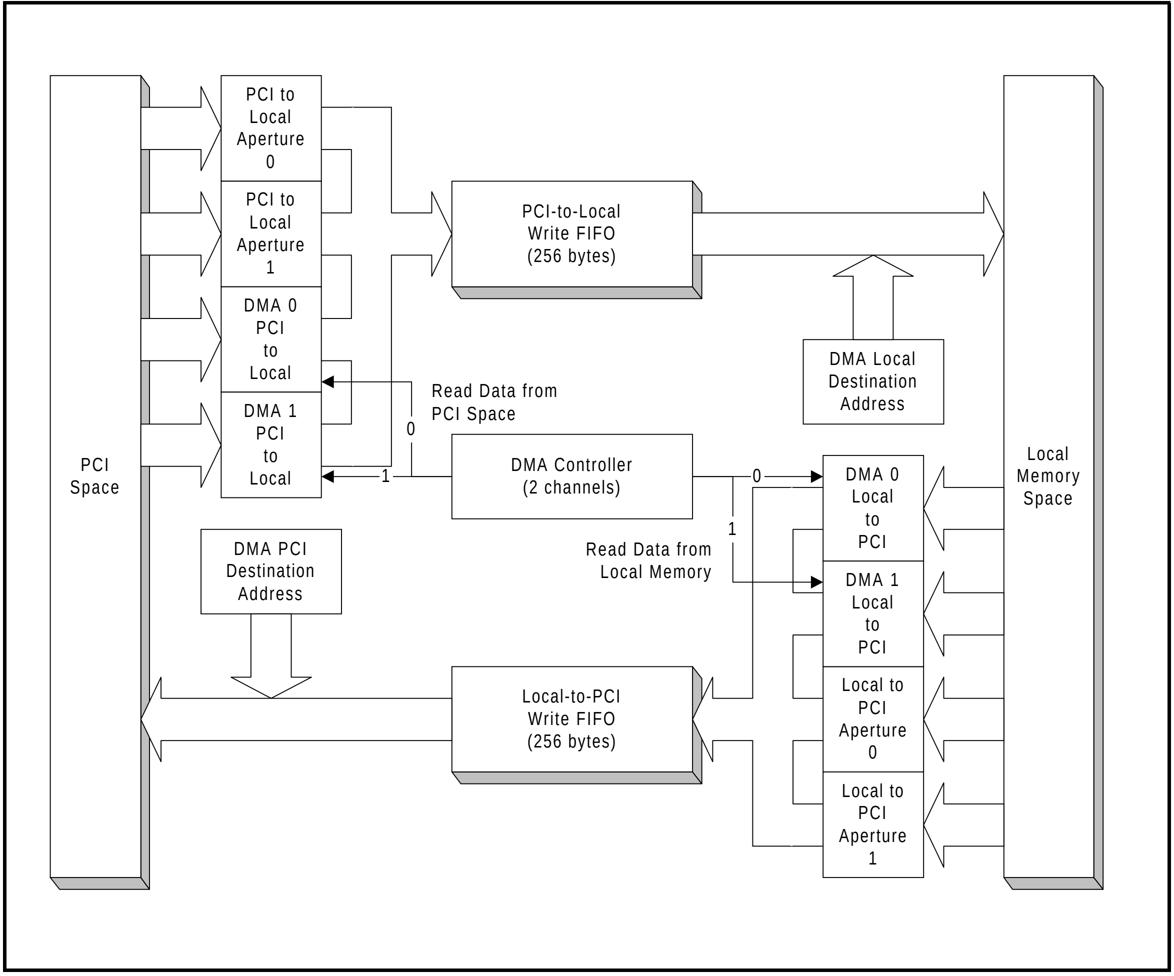
Explicit hardware DMA request inputs can be used to throttle DMA transfers. This is provided by the DREQ_EN bits in the DMA_LENGTHx registers which use $\overline{\text{INTC}}$ and/or $\overline{\text{INTD}}$ as active low DMA request inputs. Hardware throttling works by allowing the source data of a DMA transfer to be loaded only when the external $\overline{\text{DREQx}}$ pin is asserted. The act of reading or writing local memory can be used as the DMA acknowledge.

Since demand mode throttling works by gating the loading of source data, it is important that the when the local bus is the destination, it is capable of taking whatever could remain in the internal FIFO of the EPC if wait states via READY are to be avoided. In this scinerio, up to 32 words of data could be sitting in the

6.1.9 DMA Interrupts

Each DMA Controller channel will generate an interrupt whenever the transfer count reaches zero and the CHAIN bit in the corresponding DMA_CSRx register is zero (no further chains to complete). There are separate requests from each channel. The DMA interrupt requests are routed to both the Local interrupt control logic, and the PCI interrupt control logic. The DMA interrupt requests are latched by both the PCI interrupt status register and the Local interrupt status register and must be cleared independently in both status registers.

Figure 11: Write FIFO Usage by the DMA Controller



6.2 PROGRAMMING THE DMA CONTROLLER

The DMA Controller is programmed through two sets of registers, one for each of the two channels. The DMA Controller registers are accessible from both the Local and PCI sides of the EPC.

6.2.1 Setting the Starting Addresses

Two starting addresses must be initialized before DMA transfers can begin: the Local memory start address and the PCI memory start address.

The Local memory starting address is programmed through the DMA_LOCAL_ADDR0 (or DMA_LOCAL_ADDR1) register. The local memory address is word aligned. The EPC sets the 2 least significant bits of DMA_LOCAL_ADDRx to zero in order to force word alignment.

The PCI memory starting address is programmed through the DMA_PCI_ADDR0 (or DMA_PCI_ADDR1) register. The PCI memory address is word aligned. The EPC sets the 2 least significant bits of DMA_PCI_ADDRx to zero in order to force word alignment. The DMA Controller can only generate PCI Read and PCI Write commands on the PCI bus.

Both address registers are automatically incremented after each word is transferred. These registers may be read at any time to determine the current value of the pointers. The address registers may also be written to at any time, however, the programmer should check the “In Progress” bit (in the DMA_CSRx register) before modifying the pointers of a DMA operation in progress. Failure to do so could result in undefined bridge operation.

As previously mentioned, the DMA Controller is unrelated to the data transfer registers. The address translation, endian conversion, and prefetching options for the data transfer apertures will not affect DMA operation, *even if the DMA controller is performing a transfer involving memory locations that fall within a data transfer aperture.*

6.2.2 Setting the Transfer Count

The transfer count is stored in the DMA_LENGTHx register. The transfer count is in 32-bit words. The maximum value for the transfer count is 1 megaword or 4 megabytes (longer transfers may be performed using block chaining).

The transfer count is decremented after each word of data is transferred. The DMA_LENGTHx registers may be read at any time to determine the current value transfer count. The transfer count registers may also be written to at any time, however, the programmer should check the “In Progress" bit (DMA_IPR bit in the DMA_CSRx register) before modifying the transfer count for a DMA operation in progress. It is possible to halt DMA operation immediately, by setting the transfer count of a DMA process in progress to zero.

6.2.3 Setting the Transfer Direction

The DMA transfer direction - PCI-to-Local or Local-to-PCI - is set by the DIRECTION bit in the DMA_CSRx register. *This bit must not be changed while a DMA process is running* (as indicated by the state of the DMA_IPR bit for each channel.) Changing the DIRECTION bit while a DMA process is running will result in undefined bridge operation.

6.2.4 Byte Order Conversion

Each DMA channel can convert data byte order on-the-fly. The SWAP bits in the DMA_CSRx register control the three conversion options: 8-bit, 16-bit, and 32-bit. Byte order conversion by the DMA Controller is identical to that performed by the data transfer apertures.

6.2.5 Using DMA Block Chaining

A “linked list” of DMA block descriptors must be set up in local memory before initiating a chained DMA transfer. The individual descriptors do not need to occupy contiguous locations in local memory, since each descriptor includes an absolute pointer to the next (in the DMA_CTLB_ADDRx register). The address of the *second* descriptor in the chain is written to the DMA_CTLB_ADDRx register for the corresponding DMA channel. The parameters for the first block to be transferred are written directly into the DMA channel’s address, transfer count, and control/status registers.

Each descriptor in the chain must have the CHAIN bit set in the DMA_CSRx register. The last block must have the CHAIN bit cleared, to indicate to the DMA Controller that it is to terminate the process after this block is complete. There is no limit to the number of blocks in a DMA chain, in fact “ring buffers” may be easily implemented by pointing the “last” DMA block descriptor back to the “first”.

6.2.6 Starting DMA Operation

A DMA channel begins operation when the DMA Initiate Process bit (DMA_IPR) is set in the corresponding DMA_CSRx register. This bit is automatically cleared when the transfer count expires and there are no further chains to process (i.e. CHAIN = 0). Writing a zero to DMA_IPR has no affect on operation and is ignored. Once a DMA transfer has started, the corresponding DMA registers must not be written. The only exception is the "Early Termination" outlined below.

6.2.7 Early Termination of a DMA Process

A DMA process may be terminated in advance of completion by setting the ABORT bit in the corresponding DMA_CSRx register.

Using the ABORT bit will also prevent the next chain from being fetched when chaining is enabled. The ABORT bit can also be used as a "pause" bit. Once paused, the same DMA can be finished from where it left off by simply setting the DMA_IPR bit again.

Caution: There may be a delay between when the ABORT bit is set and when the DMA_IPR bit is cleared (this is due to having source data sitting in the internal FIFO waiting to write it to the destination). Once an ABORT has been initiated, software should poll DMA_IPR and wait for it to be cleared.

6.2.8 Setting Priority Between the DMA Channels

The EPC provides a simple mechanism for setting the priority between the two DMA channels. A PRIORITY bit is provided in each channel’s DMA_CSRx register. Table 9 describes the priority options that are based on the state of these bits.

Table 9: DMA Channel Priority Options

Channel 0 PRIORITY bit	Channel 1 PRIORITY bit	Result
0	0	**First Come, First Served**. Priority is assigned to the channel that is the first to initiate a DMA process (have its DMA_IPR bit set).
0	1	**Channel 1 Priority**. DMA channel 1 has the highest priority and will interrupt a DMA channel 0 transfer in process. DMA channel 0 will continue its process only after DMA channel 1 completes its transfer.
1	0	**Channel 0 Priority**. Same as above, except channel 0 has priority over channel 1.
1	1	**Rotating Priority**. The two channels rotate priority based on the least recently granted channel. When both channels are running a DMA process, they will alternate bus accesses.

Chapter 7 PCI Bus Interface

The EPC implements the PCI bus according to the revision 2.1 PCI Specification published by the PCI Special Interest Group. This section assumes a familiarity with the PCI bus specification and only describes performance and exception handling issues.

7.1 TARGET TRANSFERS

The EPC acts as a PCI target (slave) when it bridges a read or write access to one of the PCI-to-Local data transfer apertures. There are two basic types of target transfers: reads and writes.

7.1.1 Target Reads

The following command types fall under the category of target reads: Memory Read, Memory Read Multiple, I/O Read, Configuration Read, Memory Read Line, and Interrupt Acknowledge.

Upon receipt of a PCI-to-Local read request, the EPC will attempt to access the local bus by asserting the local bus request signal (BREQ or HOLD). While waiting for local bus access, the EPC will not return TRDY. When data becomes available, TRDY will be asserted and the data will be placed on the PCI bus.

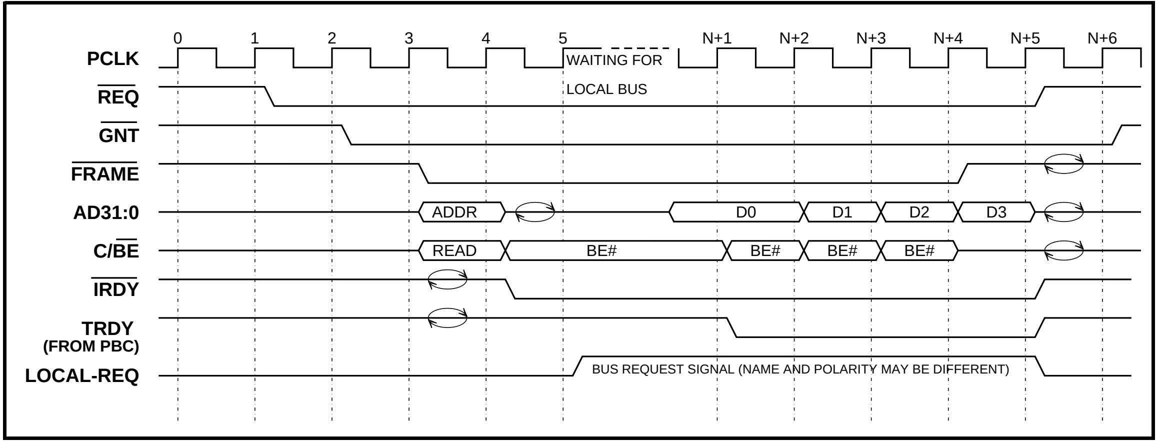
PCI 2.1 NOTE: The initial revision of the EPC (Revision A) didn't support delayed reads (although it supported the PCI 2.1 16 clock address to TRDY rule). Versions B1 and above support delayed reads, which is the PCI recommended solution for this situation.

PCI burst reads that cross a 1k byte address boundary will be broken into two smaller bursts by the EPC. This is done by issuing a PCI disconnect to the initiator as the burst crosses the 1k byte boundary.

PCI-to-Local I/O reads require one additional clock of address decoding when using the fine grain I/O PCI-to-Local aperture (see "DOS Compatibility").

Target mode reads through the PCI-to-Internal Register aperture (PC_IO_BASE) will have 3 wait-states inserted between each data cycle.

Figure 12: EPC PCI Target Mode Reads



7.1.2 Target Writes

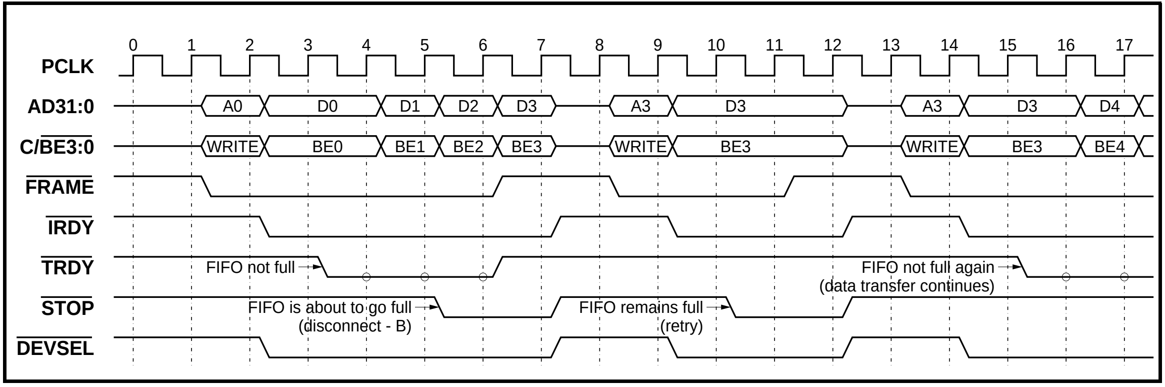
The following command types fall under the category of target writes: Memory Write, Memory Write and Invalidate, I/O Write, Configuration Write and Special Cycle.

Upon receipt of a PCI-to-Local write request, the EPC will attempt to complete the write by placing the data in the PCI-to-Local write FIFO. The EPC will complete each data phase by asserting TRDY until either the write completes (see Figure 13), or there is no room left in the write FIFO. If the PCI-to-Local write FIFO becomes full, the EPC will issue a PCI disconnect (see below).

PCI burst writes that cross a Burst boundary, as determined by PBRST_MAX in the FIFO_CFG register, will be broken into two smaller bursts by the EPC. This is done by issuing a PCI Disconnect to the initiator as the burst crosses the Burst boundary.

PCI-to-Local I/O writes require one additional clock of address decoding when using the fine grain I/O PCI-to-Local aperture (see “DOS Compatibility”).

Figure 13: EPC PCI Target Mode Aperture Writes



7.1.3 PCI Exceptions During EPC Target Cycles

The EPC only generates recoverable exceptions when acting as a PCI target.

7.1.3.1 Recoverable Exception: Target Disconnect

The EPC will generate a PCI Disconnect under the following conditions:

- A burst in progress completely fills the PCI-to-Local write FIFO.
- A PCI burst is attempted to a PCI-to-Local aperture that is disabled for bursting.
- A burst in progress crosses a 1k byte burst boundary, or exceeds the programmed maximum burst length for the aperture (as determined by the PBRST_MAX field in FIFO_CFG).
- A burst in progress is pre-empted.
- The time between two $\overline{\text{TRDY}}$ assertions in a burst exceeds 8 PCI clocks.

7.1.3.2 Recoverable Exception: Target Retry

The EPC will generate a PCI Retry under the following conditions:

- A PCI write is attempted to an already full PCI-to-Local write FIFO.
- A PCI read of Local space exceeds 16 clocks between DEVSEL and the first $\overline{\text{TRDY}}$. This is typically caused by the inability of the EPC to gain access to the local bus.
- A PCI access is attempted to the Local space while the local master is attempting a PCI access through the EPC. The EPC issues a Retry to prevent a possible deadlock.
- A PCI access is attempted while the EPC is initializing from the serial EEPROM.
- Posted reads are enabled (Bit 15 in PCI_MAPx is clear) and a read to a location not previously prefetched is performed.

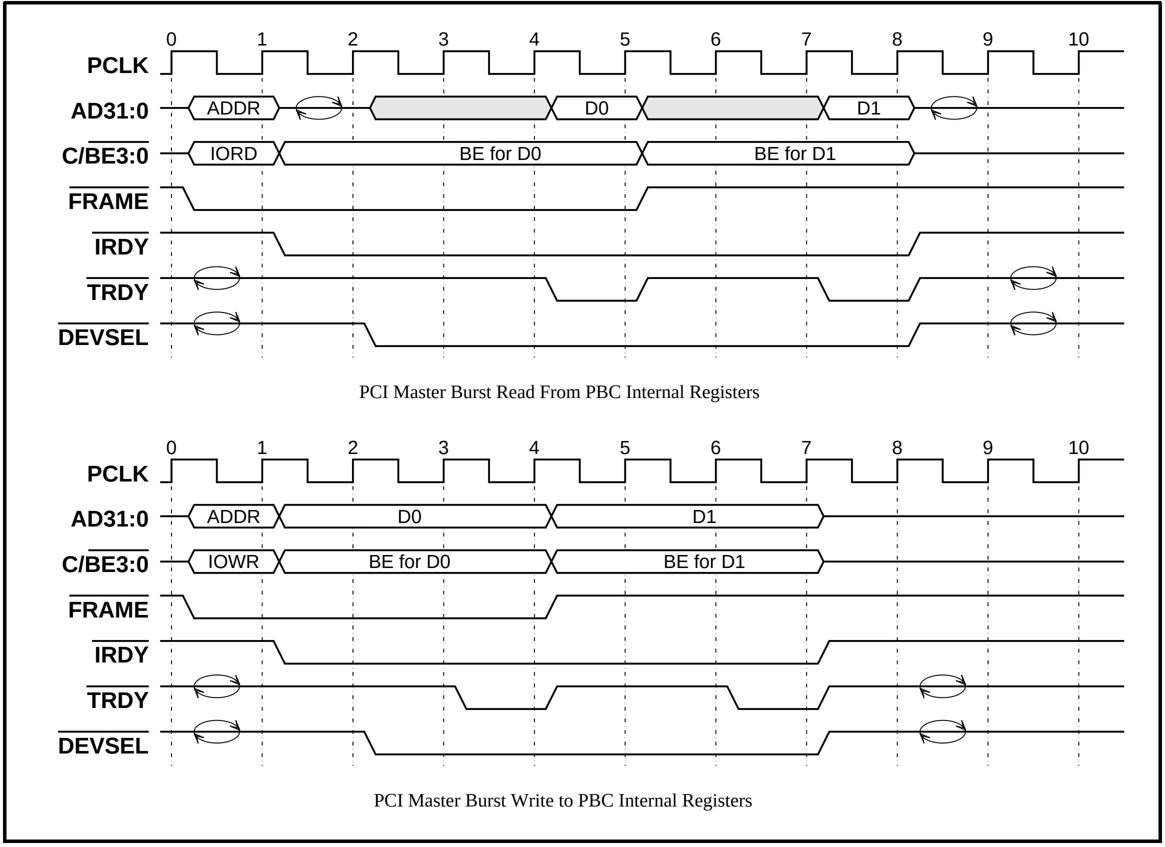
7.1.4 PCI Access of EPC Internal Registers

The internal registers of the EPC can be accessed in two ways from the PCI bus:

- Through PCI "Configuration Space" when the IDSEL input is high during the address phase and a configuration cycle type is identified on the C/$\overline{\text{BE[3:0]}}$ signals.
- Through the PCI_IO_BASE base register address as either I/O or memory.

The following waveforms illustrate the timing of a PCI master access to the internal registers.

Figure 14: PCI Read and Write of Internal EPC Registers



PCI Master Burst Read From PBC Internal Registers

PCI Master Burst Write to PBC Internal Registers

7.2 INITIATOR TRANSFERS

The EPC acts as a PCI initiator (master) when it bridges a read or write access to one of the Local-to-PCI data transfer apertures. There are two basic types of initiator transfers: reads and writes.

7.2.1 Initiator Reads

The following command types fall under the category of initiator reads: Memory Read, Memory Read Multiple, I/O Read, Configuration Read, Memory Read Line, and Interrupt Acknowledge.

The EPC will attempt to perform the fastest PCI read cycle possible ($\overline{\text{IRDY}}$ wait states are not inserted by the EPC) as shown in Figure 15 through Figure 18.

PCI reads always require a lead-off wait-state to allow for bus turnaround between the address and data phases. A PCI read cycle may be extended by slower targets not returning $\overline{\text{TRDY}}$ until the target is ready to return data.

7.2.2 Initiator Writes

The following command types fall under the category of initiator reads: Memory Write, Memory Write and Invalidate, I/O Write, Configuration Write and Special Cycle.

The EPC will attempt to perform the fastest PCI write cycle possible as shown in Figure 19 through Figure 22. Unlike PCI read cycles, PCI writes do not require bus turnaround between the address and data phases. A PCI write cycle may be extended by slower targets not returning $\overline{\text{TRDY}}$ until the target is ready to receive data.

When the LB_WR_PCI bits in FIFO_CFG register are programmed to '00', the local bus write cycle will cause a PCI bus request as soon as possible. Other settings of LB_WR_PCI will cause $\overline{\text{REQ}}$ assertion to be delayed.

Table 10: FIFO control for Local Bus Write to PCI Bus Aperture 0 and 1

LB_WR_PCI	Result
00	Assert PCI bus request immediately whenever the corresponding FIFO is not empty
01	Reserved
10	Assert PCI bus request whenever the Local bus to PCI corresponding FIFO has 3 or more words of data pending
11	Assert PCI bus request whenever the Local bus to PCI corresponding FIFO has 3 or more words of data pending *or* the FIFO is not empty and the local bus master ends a burst write cycle to the FIFO

Figure 15: V350EPC (960 mode) Initiated PCI Read Cycle

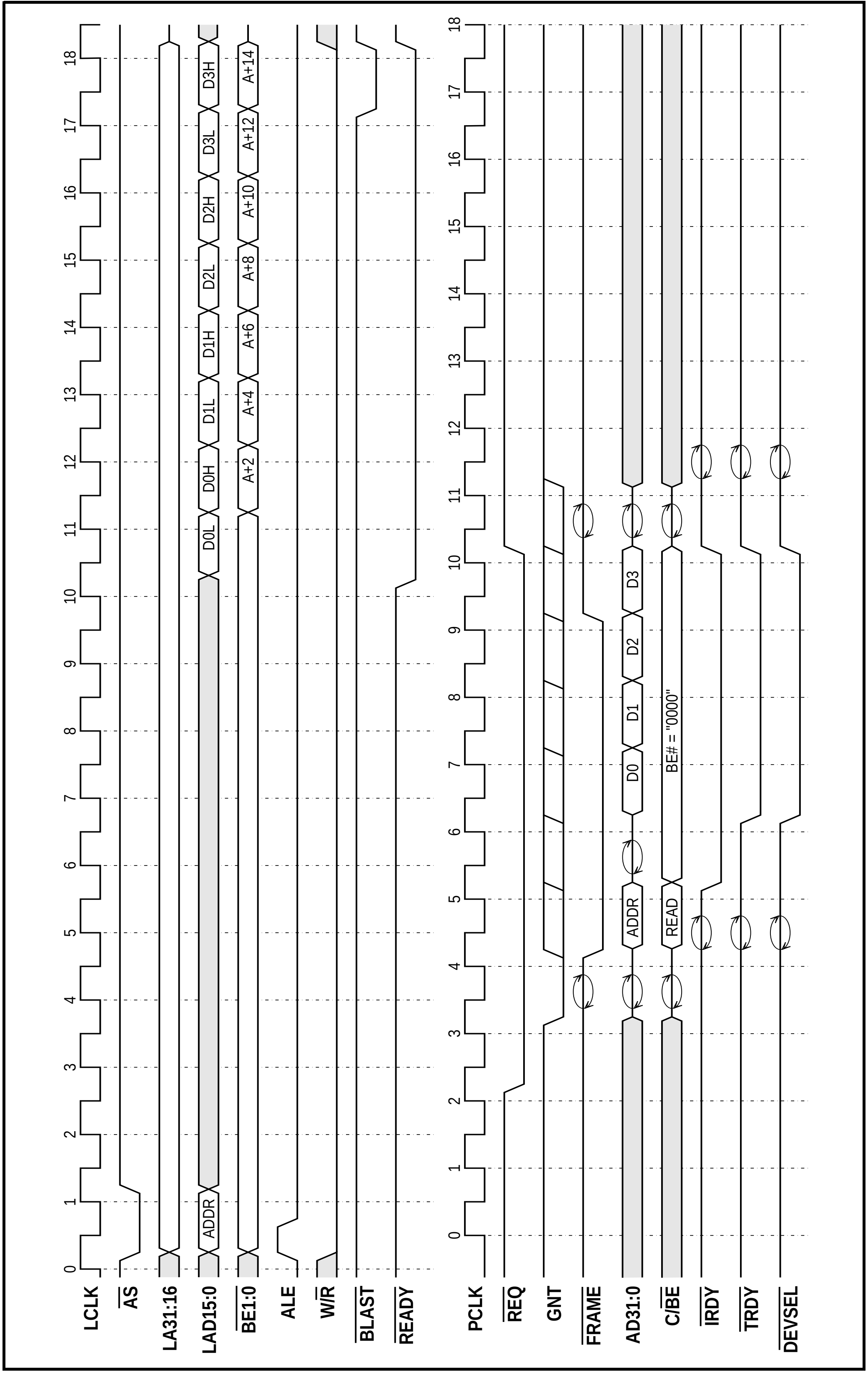


Figure 16: V350EPC (961 mode) Initiated PCI Read Cycle

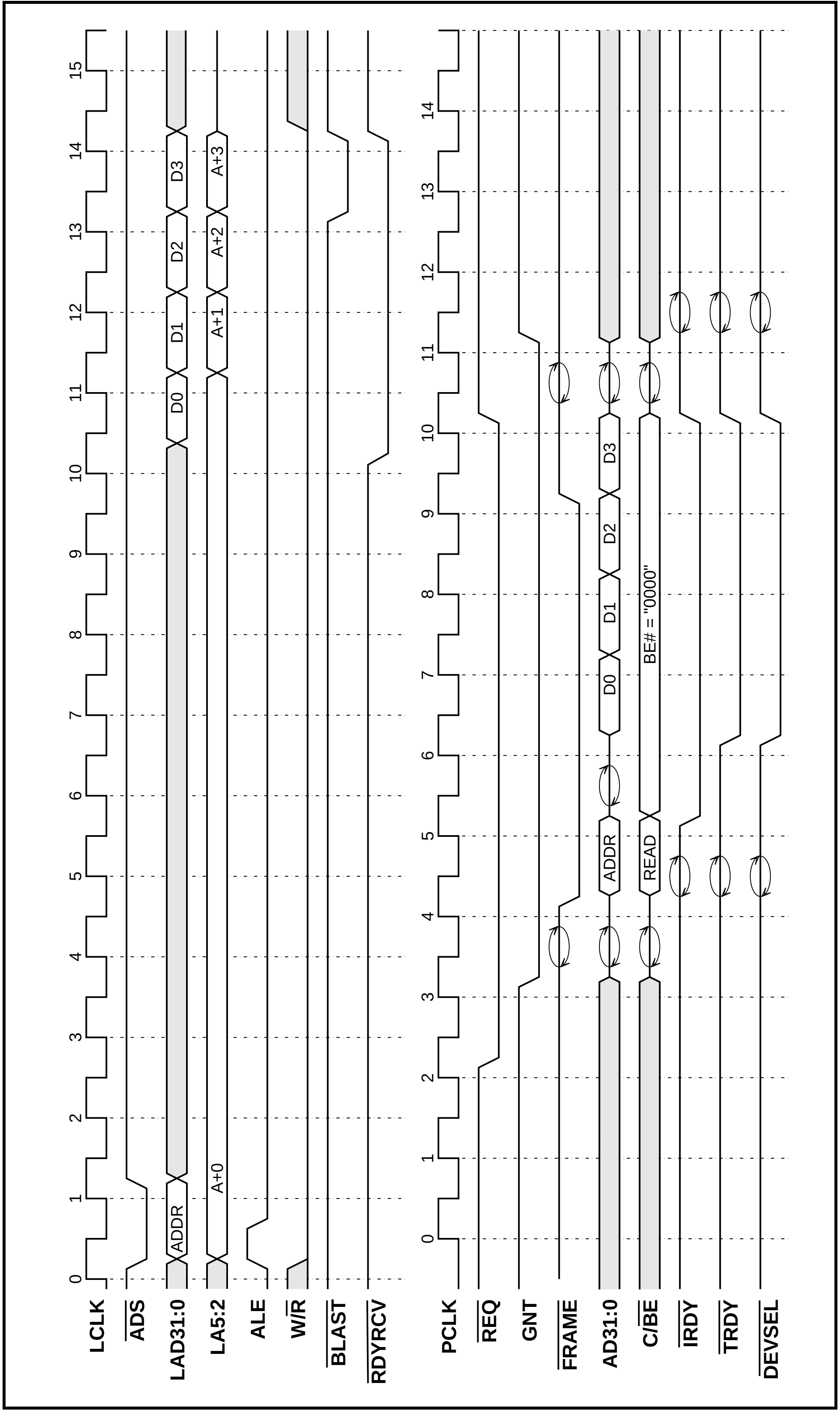


Figure 17: V360EPC (962 mode) Initiated PCI Read Cycle

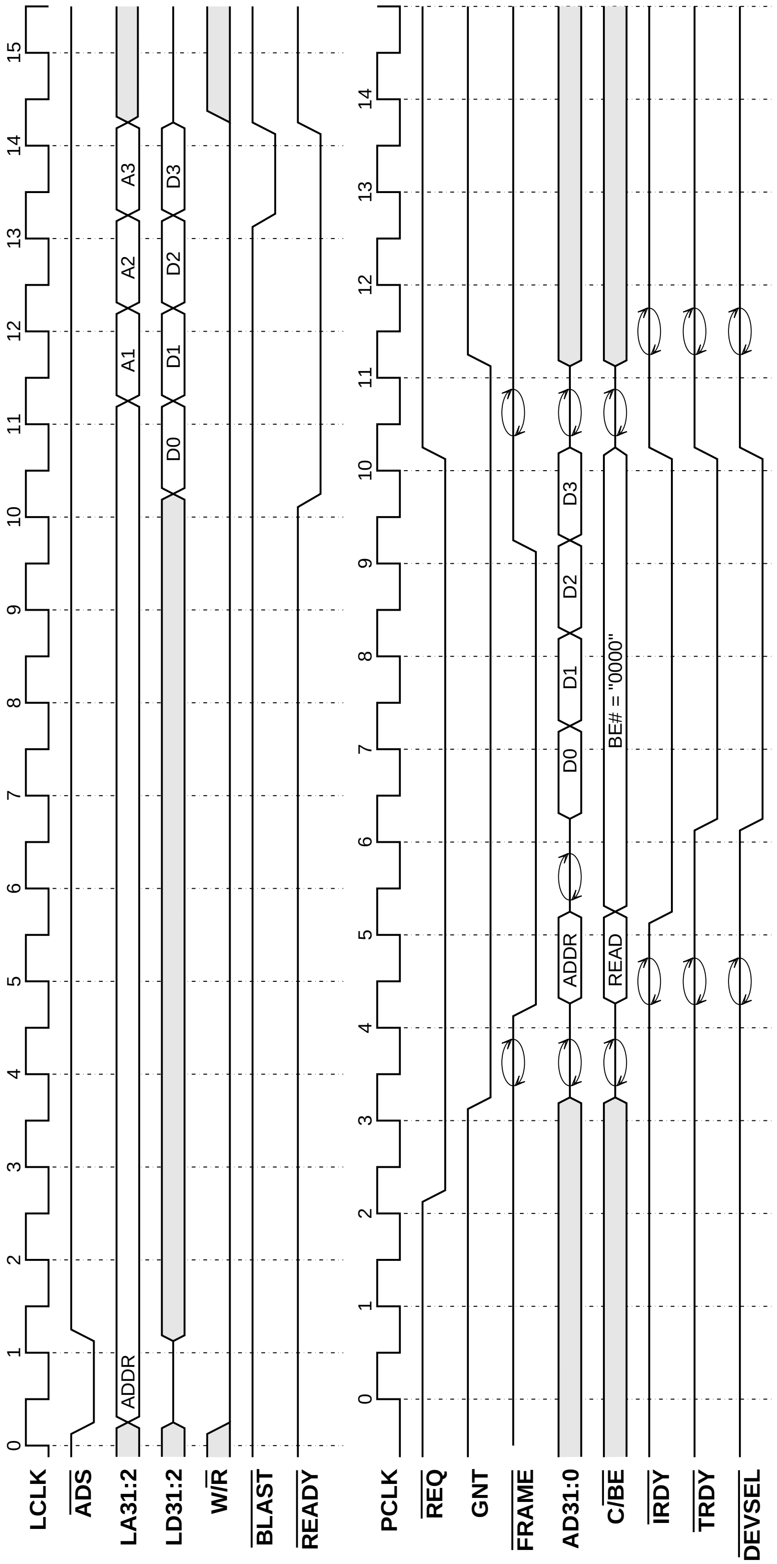


Figure 18: V360EPC (292 mode) Initiated PCI Read Cycle

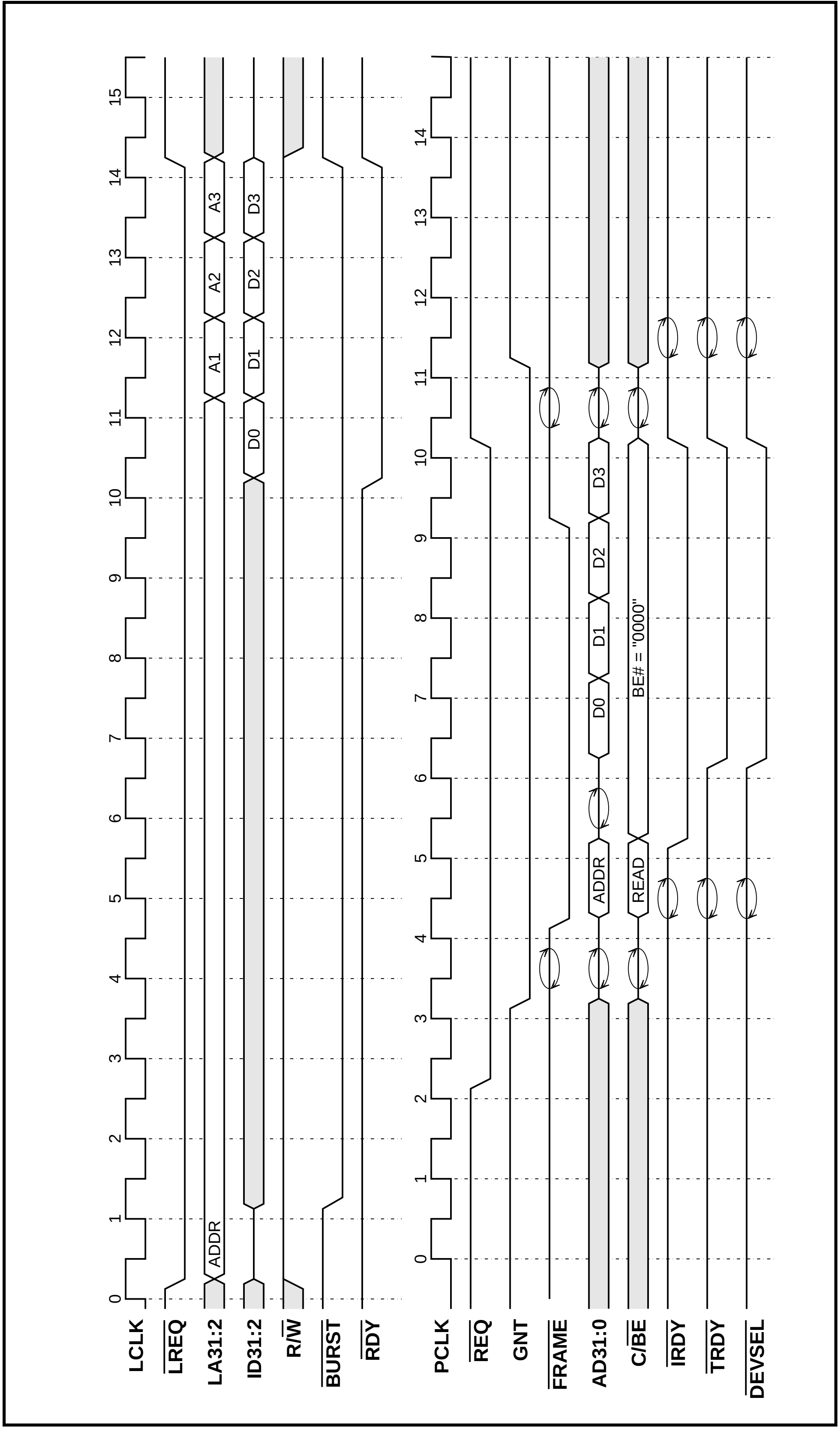


Figure 19: V350EPC (960 mode) Initiated PCI Write Cycle

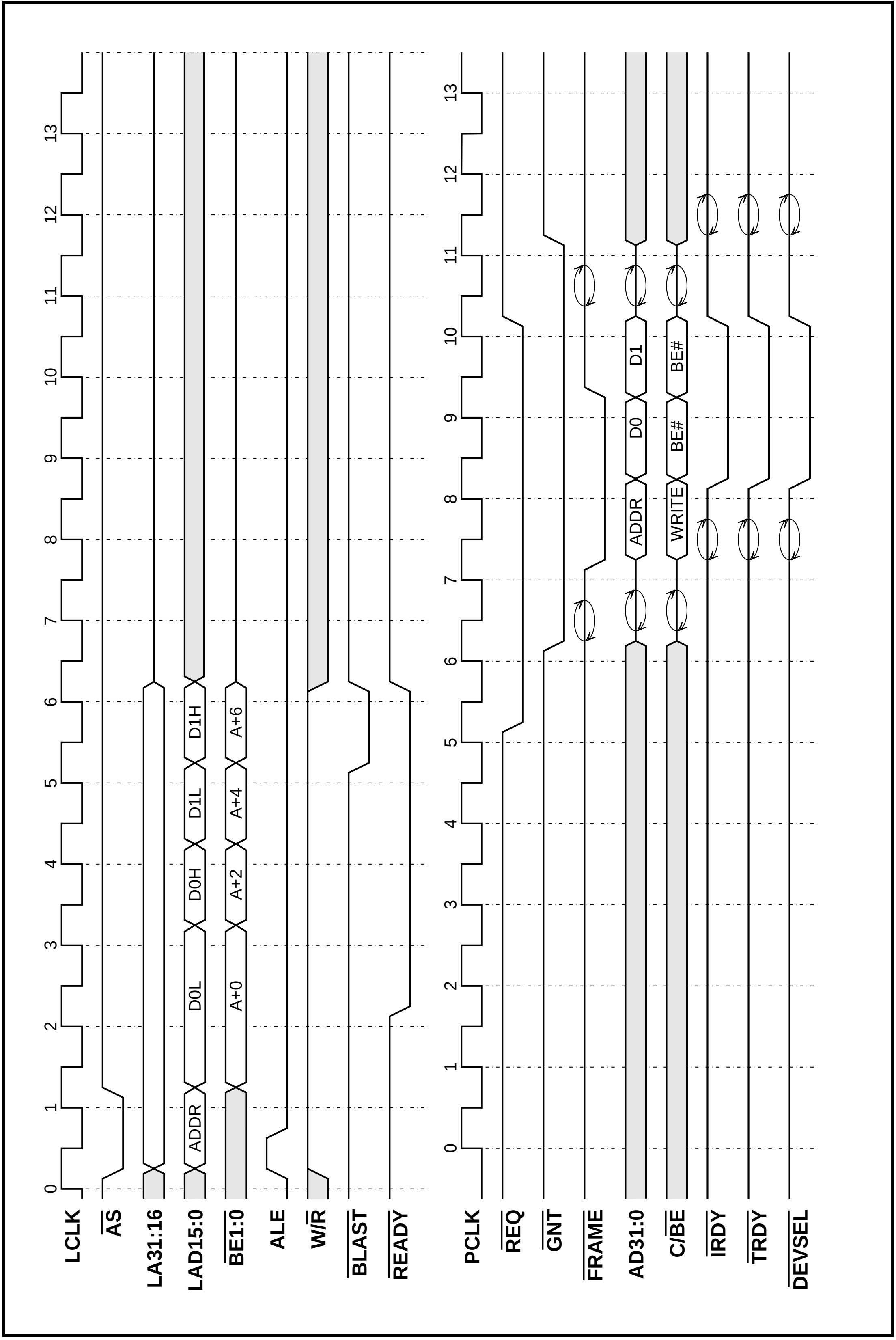


Figure 20: V350EPC (961 mode) Initiated PCI Write Cycle

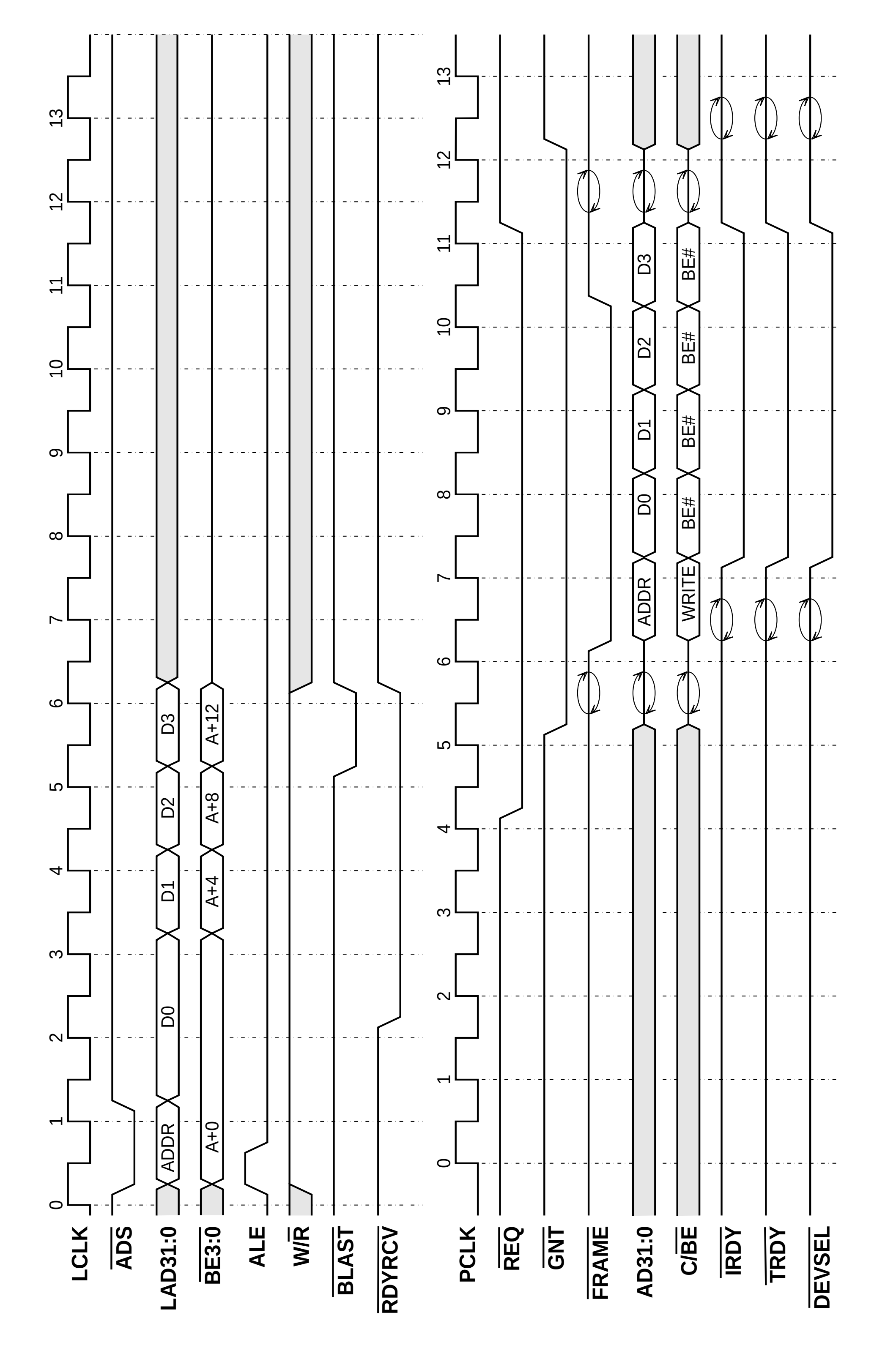


Figure 21: V360EPC (962 mode) Initiated PCI Write Cycle

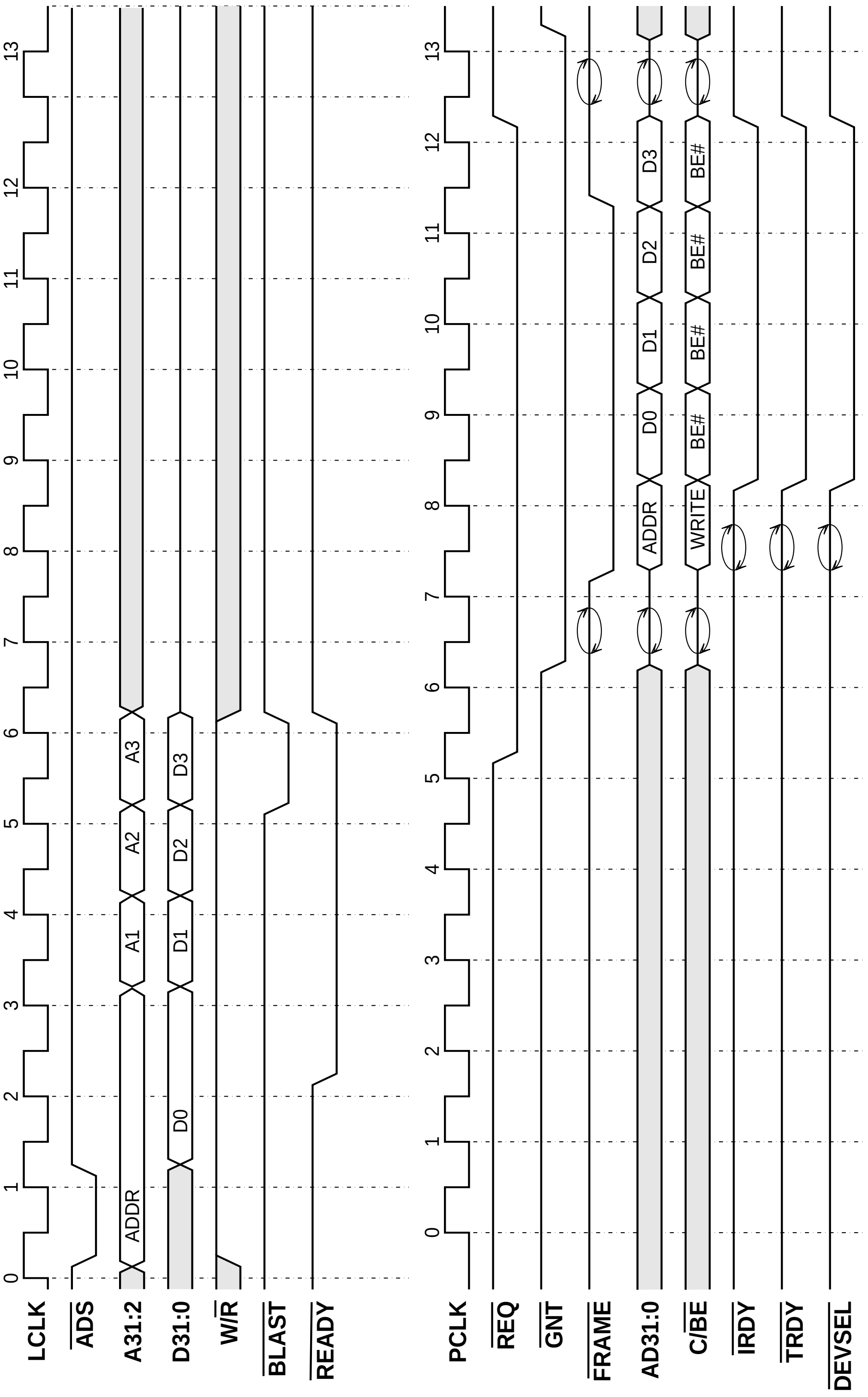
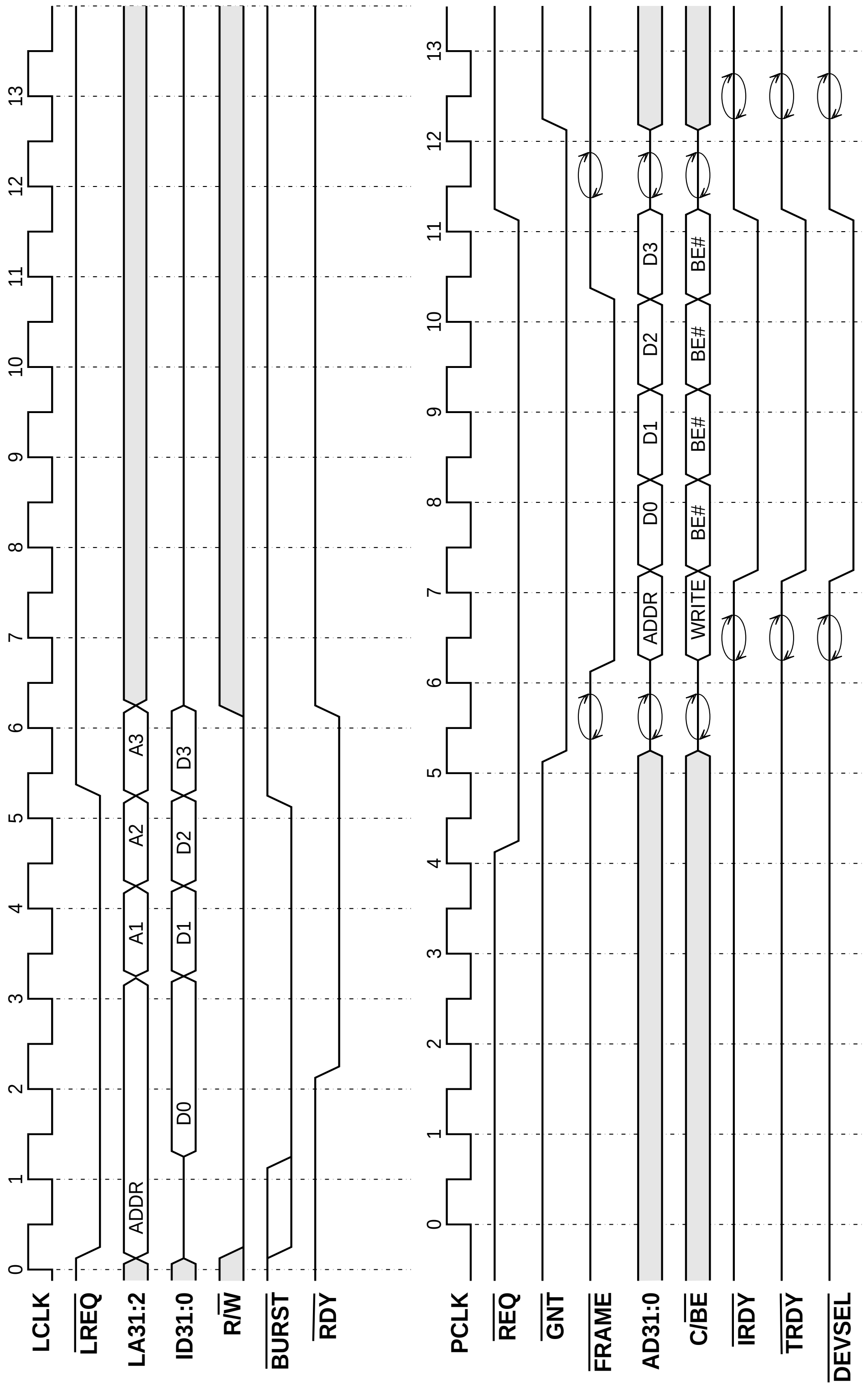


Figure 22: V360EPC (292 mode) Initiated PCI Write Cycle



7.2.3 PCI Exceptions During EPC Initiated Cycles

There are two categories of exceptions that can be generated on the PCI bus:

Fatal Exceptions: These are conditions from which recovery is not possible such as Master Abort (attempted access to non-existent device) and Target Abort (unrecoverable error within a target).

Recoverable Exceptions: These are exceptions from which recovery is guaranteed such as Disconnect and Retry. A target disconnect, for example, is used by a target to discontinue a burst that is longer than it can accommodate in a single transfer. This exception is recovered by simply starting another PCI cycle to complete the transaction.

Fatal exceptions are reported to the local processor by a processor interrupt. Recoverable exceptions are handled transparently by the EPC.

7.2.3.1 Fatal Exception: Master Abort (Reads)

A Master Abort occurs when the EPC does not receive a DEVSEL in response to a read or write attempt. This typically results from an attempt to access a non-existent device.

The EPC responds to Master Abort conditions during a read as follows:

- READY is returned to the local processor to “unlock” the local bus. The data returned is FFFFFFFFH. Optionally, a PCI read exception interrupt can be generated.
- The M_ABORT bit in the PCI_STAT register is set.

7.2.3.2 Fatal Exception: Master Abort (Writes)

Since PCI writes are posted by the EPC, it is conceivable that a Master Abort exception can occur long after the local bus write completes. In this case it makes no sense to generate a bus error on the local bus.

The EPC responds to Master Abort conditions during a write as follows:

- A PCI write exception interrupt is generated (maskable).
- The M_ABORT bit in the PCI_STAT register is set.

7.2.3.3 Fatal Exception: Target Abort (Reads)

A Target Abort occurs when the EPC receives a target abort indication from the currently addressed target. Target Abort is only used in the most extreme cases since it implies that the target is in the system (i.e. a DEVSEL is received), however it is permanently incapable of responding to the request.

The EPC responds to Target Abort conditions during a read as follows:

- READY is returned to the local processor to “unlock” the local bus. The data returned is indeterminate. Optionally, a PCI read exception interrupt can be generated.

- The T_ABORT bit in the PCI_STAT register is set.

As a target, the PCI will never generate a target abort.

7.2.3.4 Fatal Exception: Target Abort (Writes)

Since PCI writes are posted by the EPC, it is conceivable that a Target Abort exception can occur long after the local bus write completes. In this case it makes no sense to generate a bus error on the local bus.

The EPC responds to Target Abort conditions during a write as follows:

- A PCI write exception interrupt is generated (maskable).

- The T_ABORT bit in the PCI_STAT register is set.

7.2.3.5 Recoverable Exception: Target Disconnect

A target responds with a Target Disconnect when it is no longer capable of receiving data during a transaction. For example, if a target’s write buffer became full during a long burst it can issue a Target Disconnect to break the burst into smaller “pieces”. Target Disconnect informs the initiator that the burst can be restarted at the point at which the disconnect occurred (i.e. the initiator does not need to repeat the entire burst).

The EPC handles Target Disconnect transparently by simply restarting the transaction at the point at which it was “disconnected”. No errors are reported since no data is lost.

7.2.3.6 Recoverable Exception: Target Retry

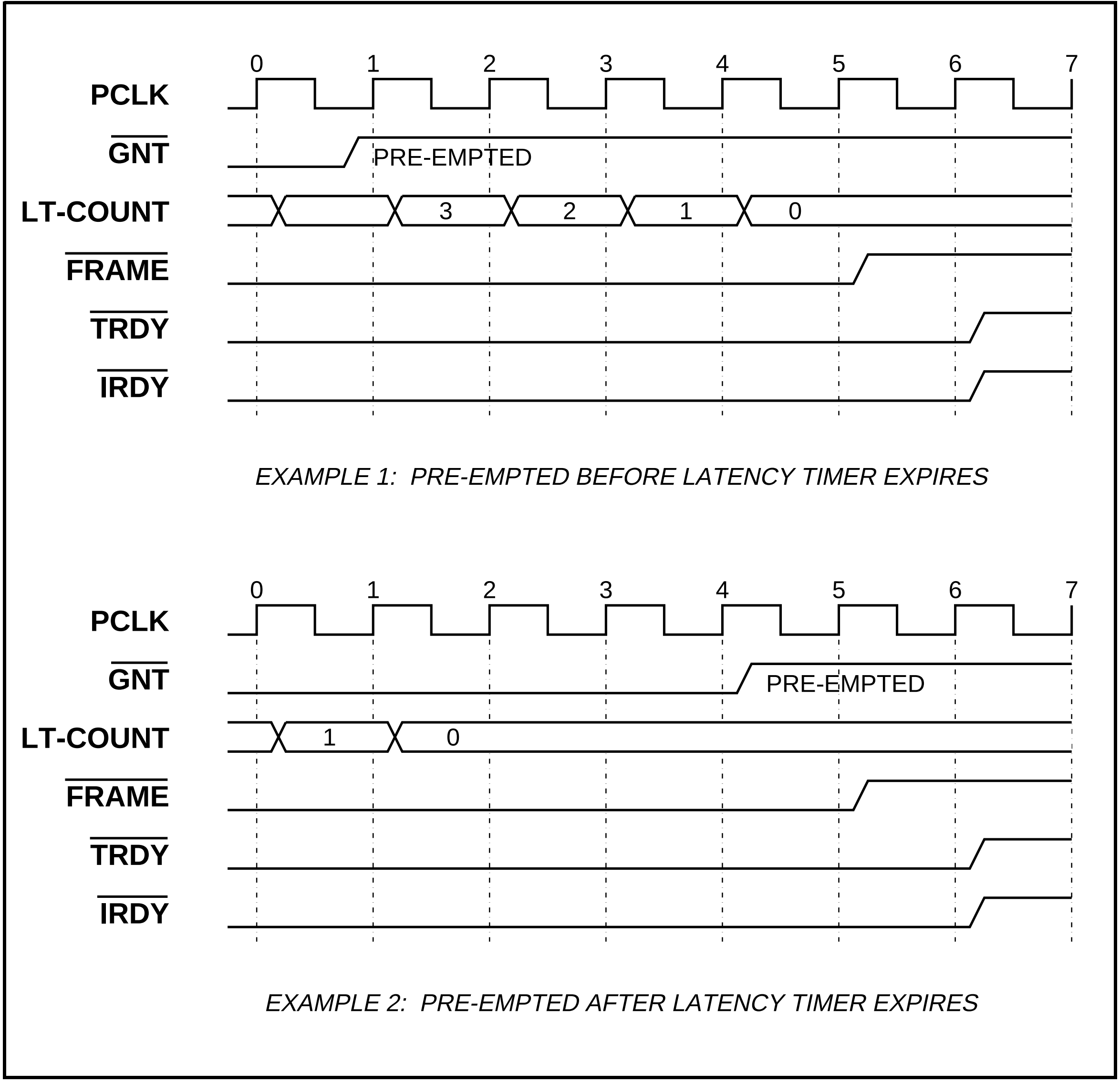
A target responds with a Target Retry when it is not capable of receiving data during a transaction. For example, if a target is currently too busy to respond to a PCI request, it can issue a Retry to tell the initiator “come back later”. Target Retry informs the initiator that the burst must be restarted from the beginning (i.e. the initiator needs to repeat the entire burst).

The EPC handles Target Retry transparently by simply restarting the transaction at the point at which it was “retried”. No errors are reported since no data is lost.

7.2.4 Initiator Pre-Emption

A PCI initiator is said to be pre-empted whenever its $\overline{\text{GNT}}$ line is deasserted during an active transfer. When the $\overline{\text{GNT}}$ is deasserted, the EPC checks its latency timer. If the latency timer has not expired, the EPC will maintain ownership of the bus until the timer reaches zero. If the latency timer has reached zero, the EPC will relinquish the PCI bus following the next data transfer. Both cases are shown in Figure 23.

Figure 23: Initiator Pre-Emption



EXAMPLE 1: PRE-EMPTED BEFORE LATENCY TIMER EXPIRES

EXAMPLE 2: PRE-EMPTED AFTER LATENCY TIMER EXPIRES

Chapter 8 Local Bus Interface

The EPC family devices are designed to be directly connected to i960®(Am29K$^{\text{TM}}$) family processors without the use of any "glue logic". The local bus protocol used by the EPC devices duplicate the bus protocol of the corresponding i960(Am29K) processor. For example, the EPC uses the same bus protocol as the i960(Am29K).

The EPC acts as both a local bus target and as a local bus master. This section of the manual describes the local bus interface for all versions of the EPC family.

8.1 TARGET MODE

The local bus interface is said to be in *target mode* when the EPC is responding to read and write requests from the local bus master (normally an i960(Am29K) family processor). There are currently three i960 processor buses supported: i960Sx (V350EPC in V960 mode), i960Jx (V350EPC in V961 mode), i960Cx/Hx (V360EPC in 962 mode) and two AMD processor buses:Am29030/40(V360EPC in 292 mode).

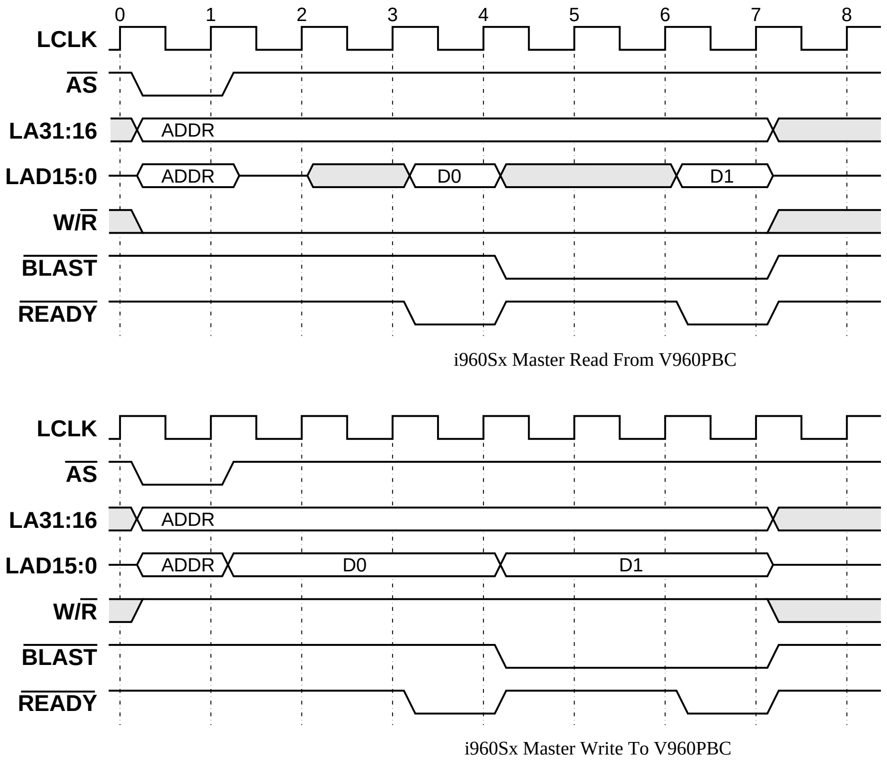
8.1.1 Local Bus CPU Configuration

Some local bus CPU devices, such as the i960Cx, have programmable bus parameters such as wait states and bus size. The EPC devices work at the largest bus width of the processor which is 32 bits for all EPC members except the V350EPC (960 mode) which is 16 bits to match the i960Sx 16 bit data width. Also, the EPC devices control the number of wait states for each access and do not rely on the processors programmable wait state generators. Therefor, for the regions that access the EPC, the CPU should be set up for 32 bit bus width (16 for the V350EPC), external ready enabled and minimum wait state settings.

8.1.2 Local Reads and Writes to Internal Registers

Reads and writes to the EPC's internal registers occur through the Local-to-Internal Register aperture defined by the LB_IO_BASE register. Internal register read/writes always take 2 wait-states. Control of the local bus $\overline{\text{READY}}$ signal is provided by the EPC. Burst accesses are permitted to internal registers. Figure 24 through Figure 27 show a local-to-internal register read/write access for each of the EPC components.

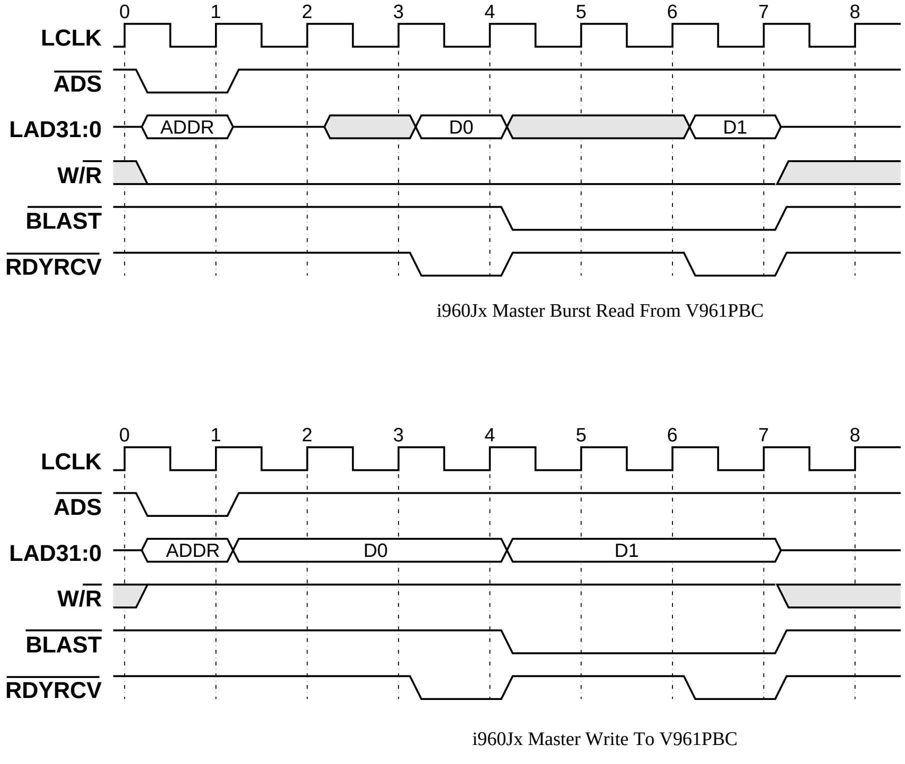
Figure 24: Local Master Read/Write to Internal V350EPC Registers (i960Sx Bus)



i960Sx Master Read From V960PBC

i960Sx Master Write To V960PBC

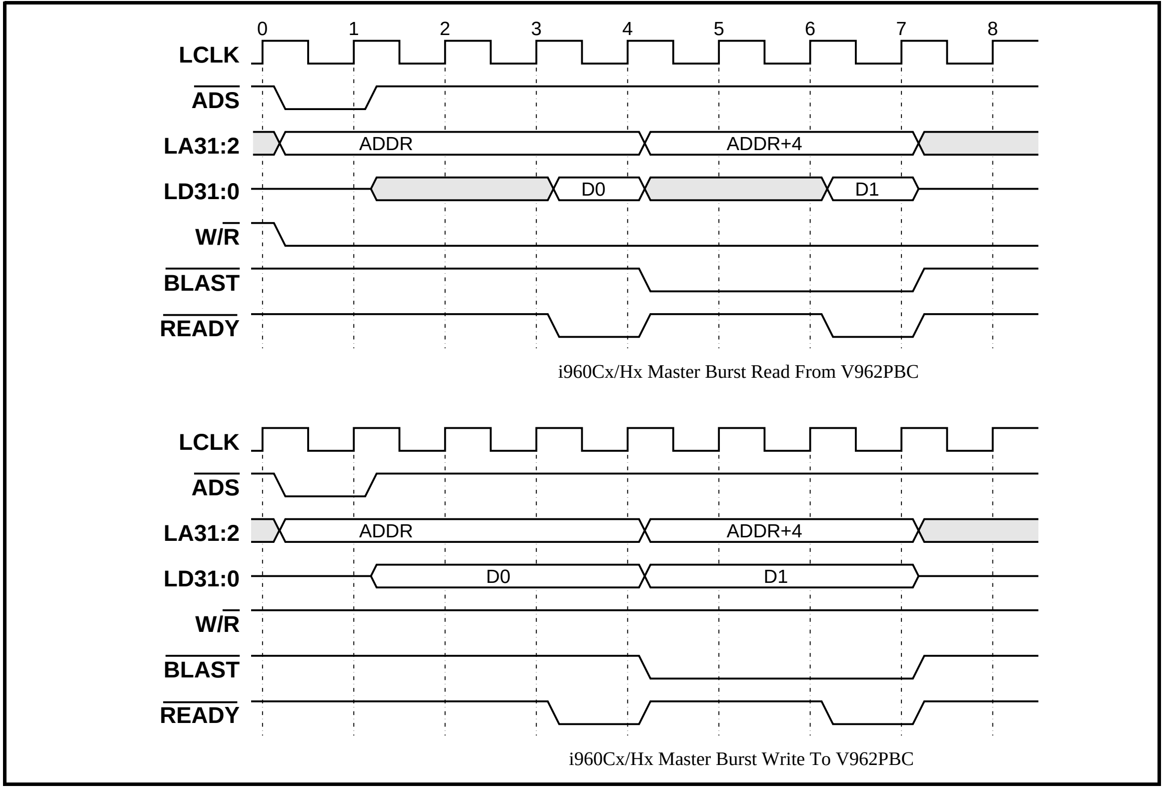
Figure 25: Local Master Read/Write to Internal V350EPC Registers (i960Jx Bus)



i960Jx Master Burst Read From V961PBC

i960Jx Master Write To V961PBC

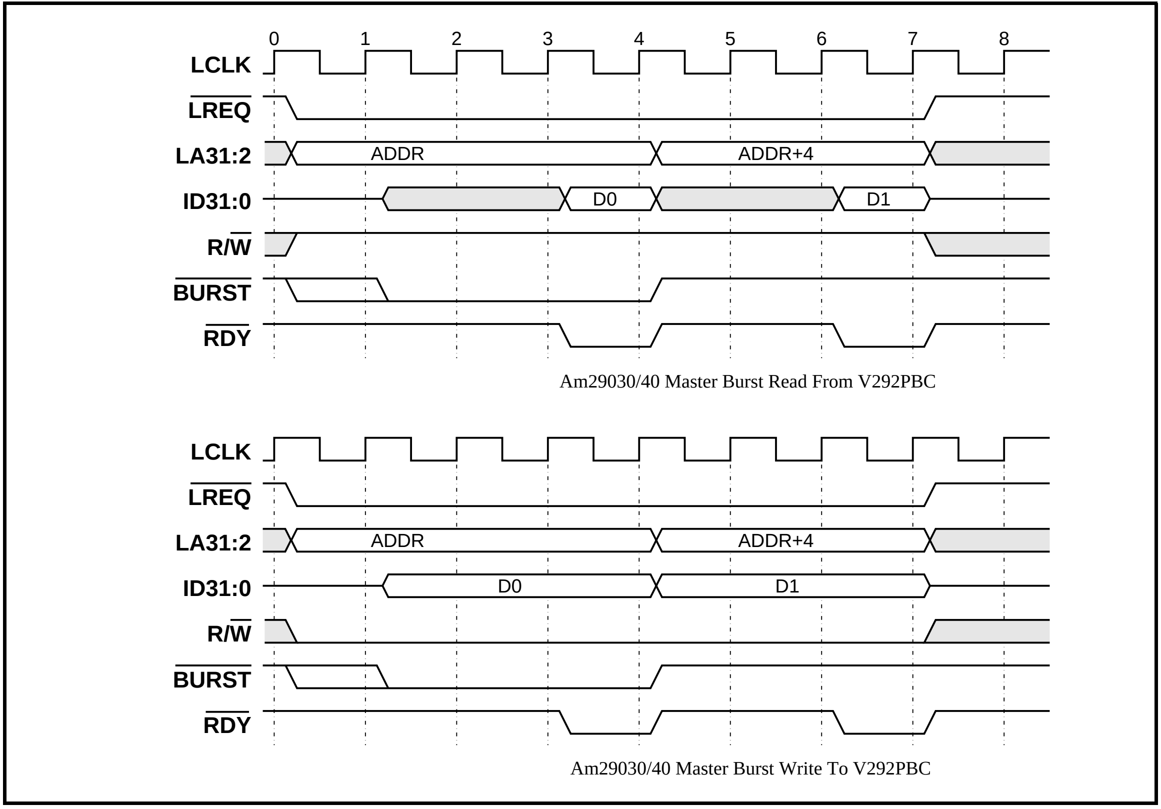
Figure 26: Local Master Read/Write to Internal V360EPC Registers (i960Cx/Hx Bus)



i960Cx/Hx Master Burst Read From V962PBC

i960Cx/Hx Master Burst Write To V962PBC

Figure 27: Local Master Read/Write to Internal V360EPC Registers (Am29030/40 Bus)



Am29030/40 Master Burst Read From V292PBC

Am29030/40 Master Burst Write To V292PBC

8.1.3 Local Read from Local-to-PCI Apertures

Local reads from PCI space take place through the Local-to-PCI apertures. The initial read to any aperture will incur the full penalty of waiting for the corresponding read to complete on the PCI bus. The local bus is held NOT READY until the first datum is available from the PCI bus. Subsequent reads will complete as fast as is possible based on the following:

- How many words, if any, are buffered in the read FIFO.
- Whether or not prefetching is turned on for the aperture.
- How large a delta there is in the operating frequencies of the PCI and local bus.

The fastest local bus reads will complete is one wait-state for address-to-data and then zero wait-states for data to data. Zero wait-state reads will only occur when the read in progress is “draining” prefetched data from a Local-to-PCI read FIFO.

The EPC will attempt to perform the fastest local read cycle possible as shown in Figure 15 through Figure 22.

8.1.4 Local Write to Local-to-PCI Apertures

Local writes to PCI space take place through the Local-to-PCI apertures. Writes are posted at zero wait-state until no room is left in the write FIFO. Attempts to write to a full write FIFO will result in the local bus being held NOT READY until room becomes available. Figure 32 through Figure 35 show the fastest local bus writes to the Local-to-PCI write FIFO.

Figure 28: V350EPC (960 mode) PCI Posted Read from Local Bus

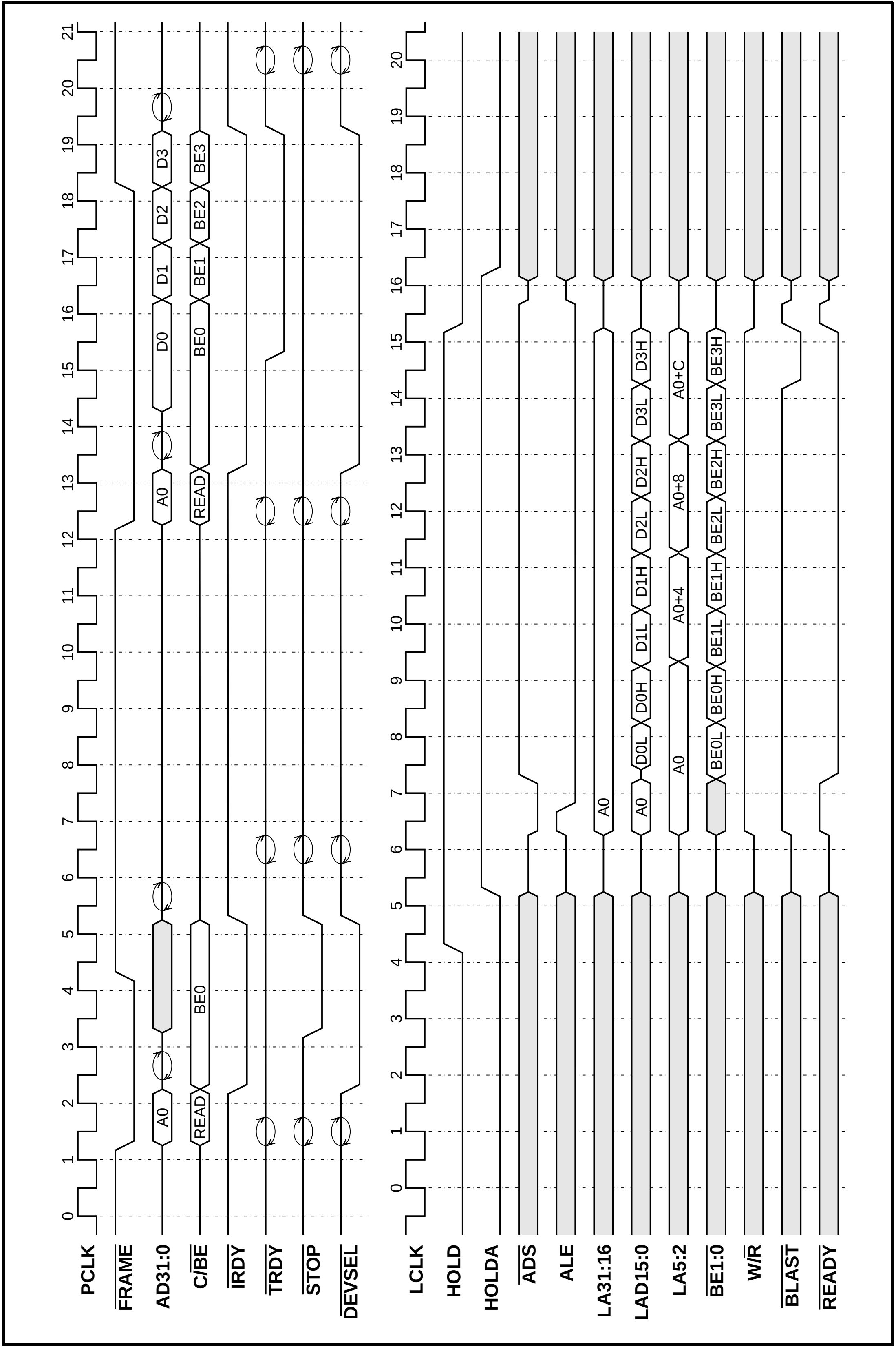
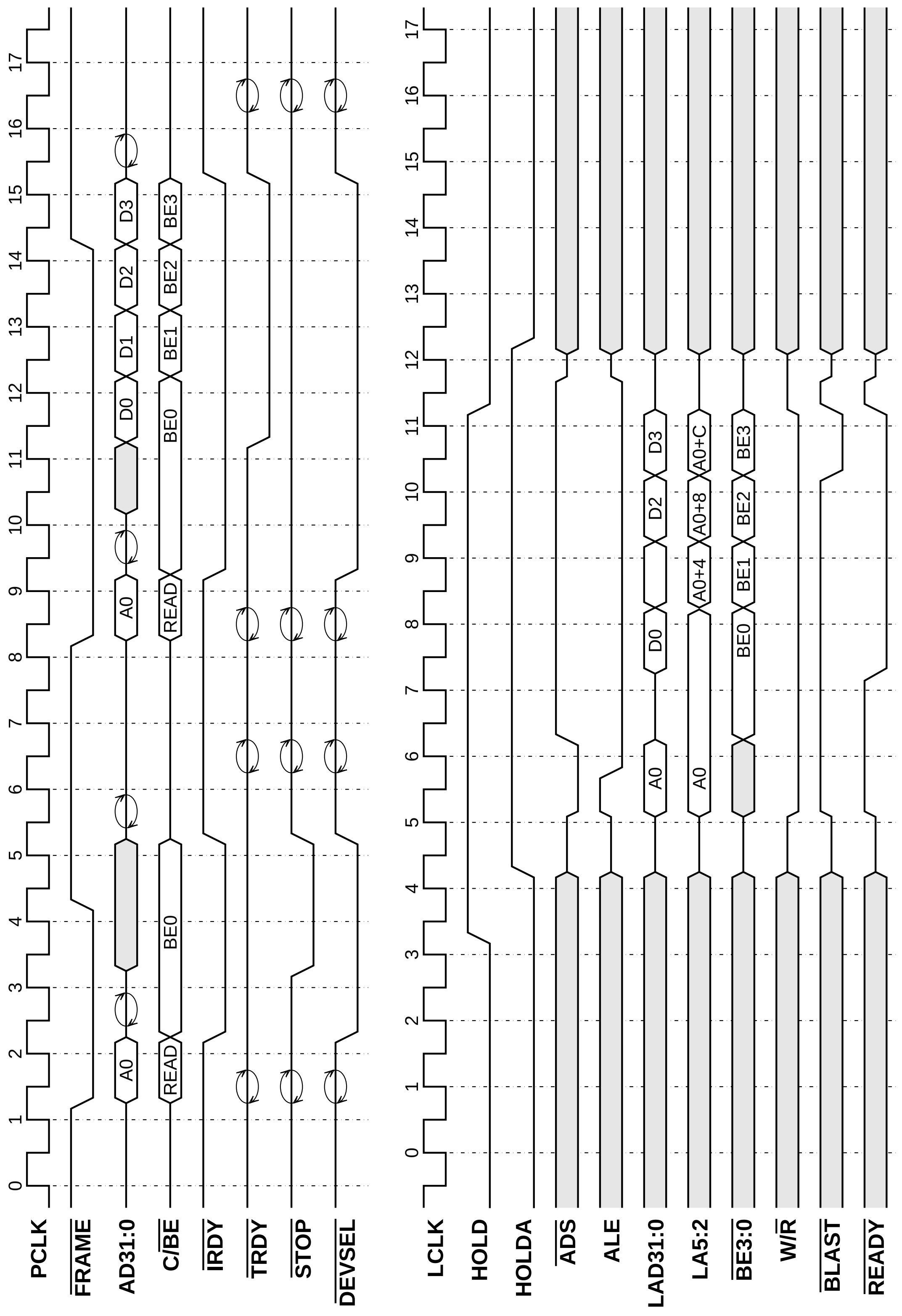


Figure 29: V350EPC (961 mode) PCI Posted Read from Local Bus



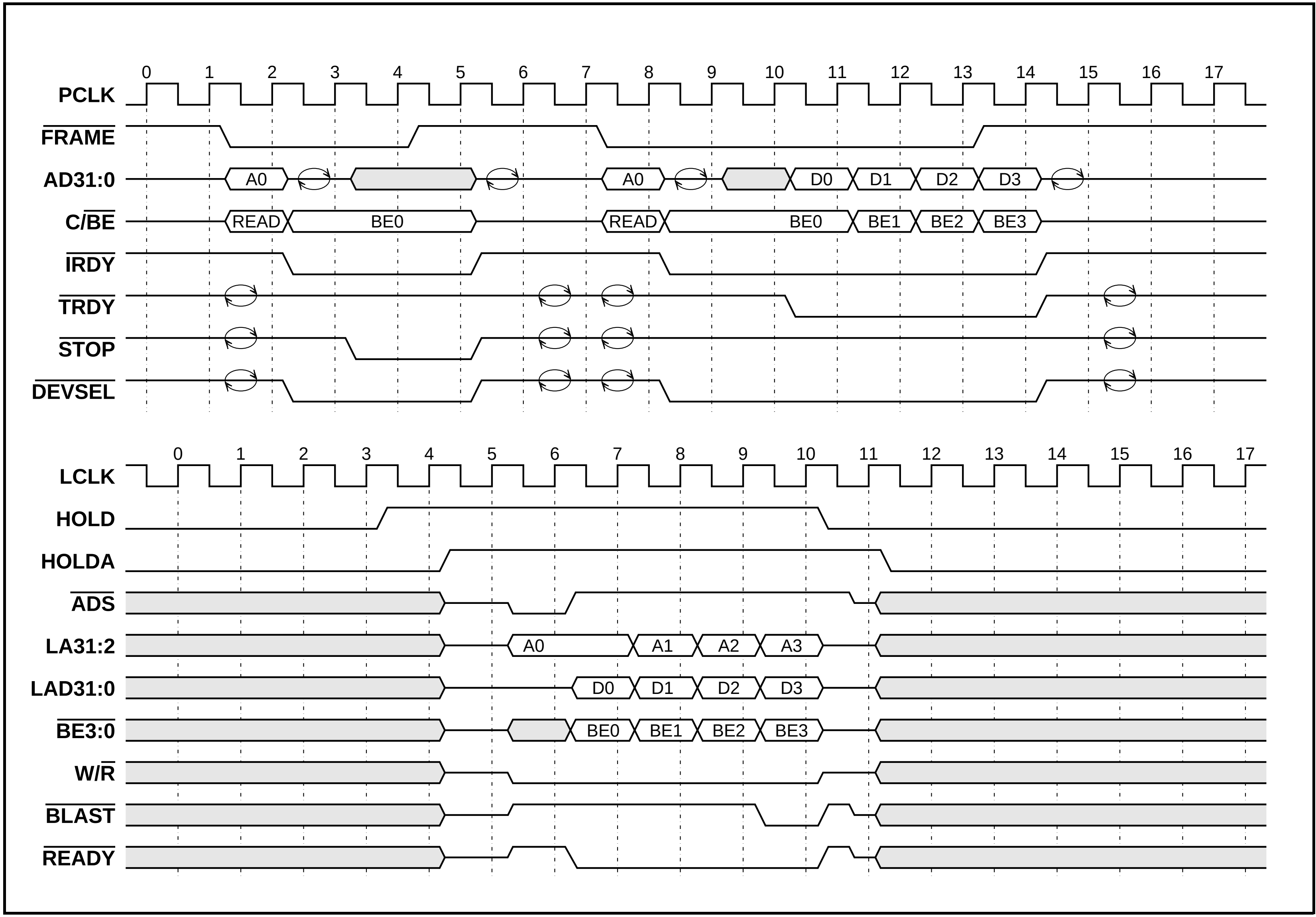
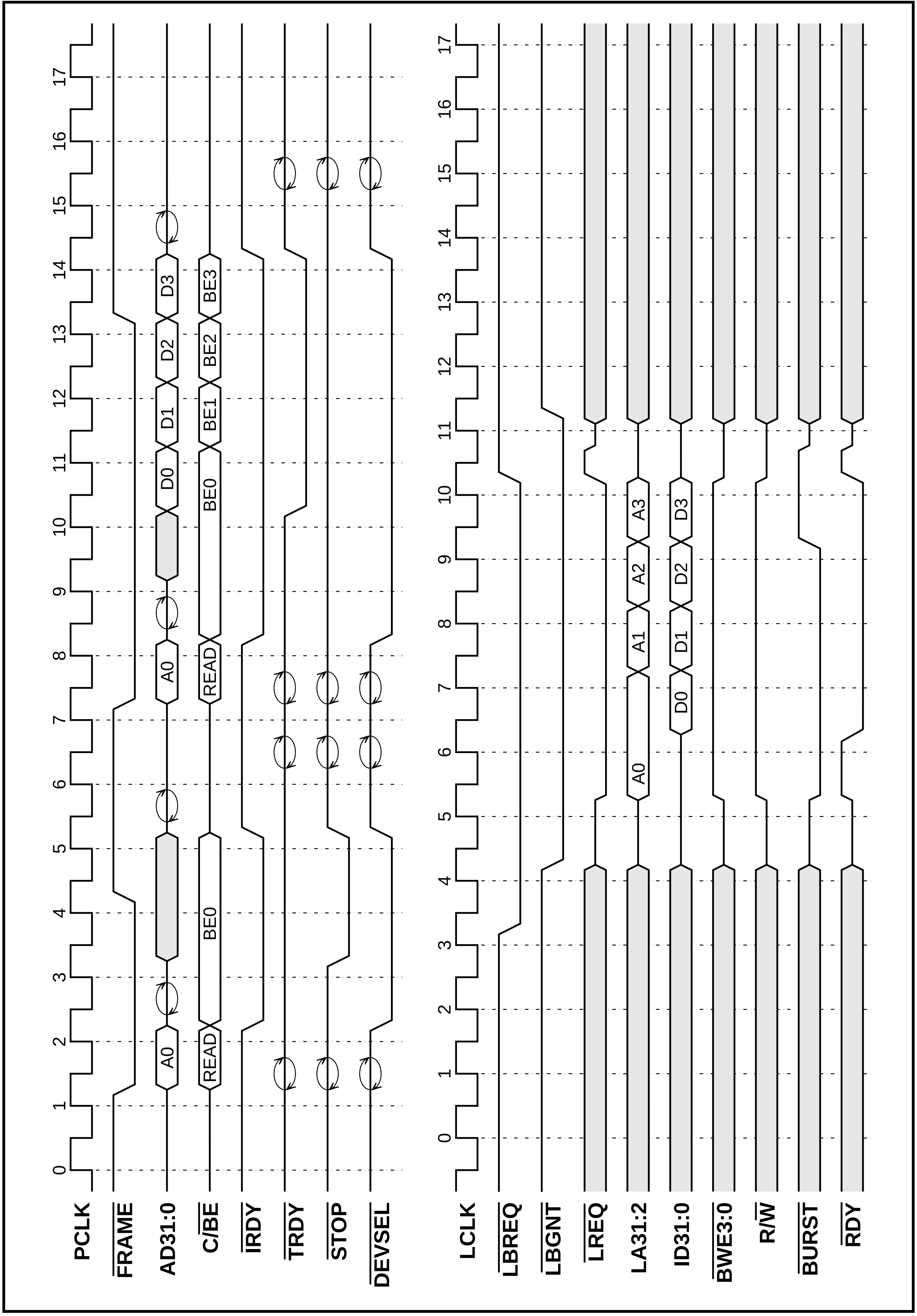


Figure 30: V360EPC (962 mode) PCI Posted Read from Local Bus

Figure 31: V360EPC (292 mode) PCI Posted Read from Local Bus



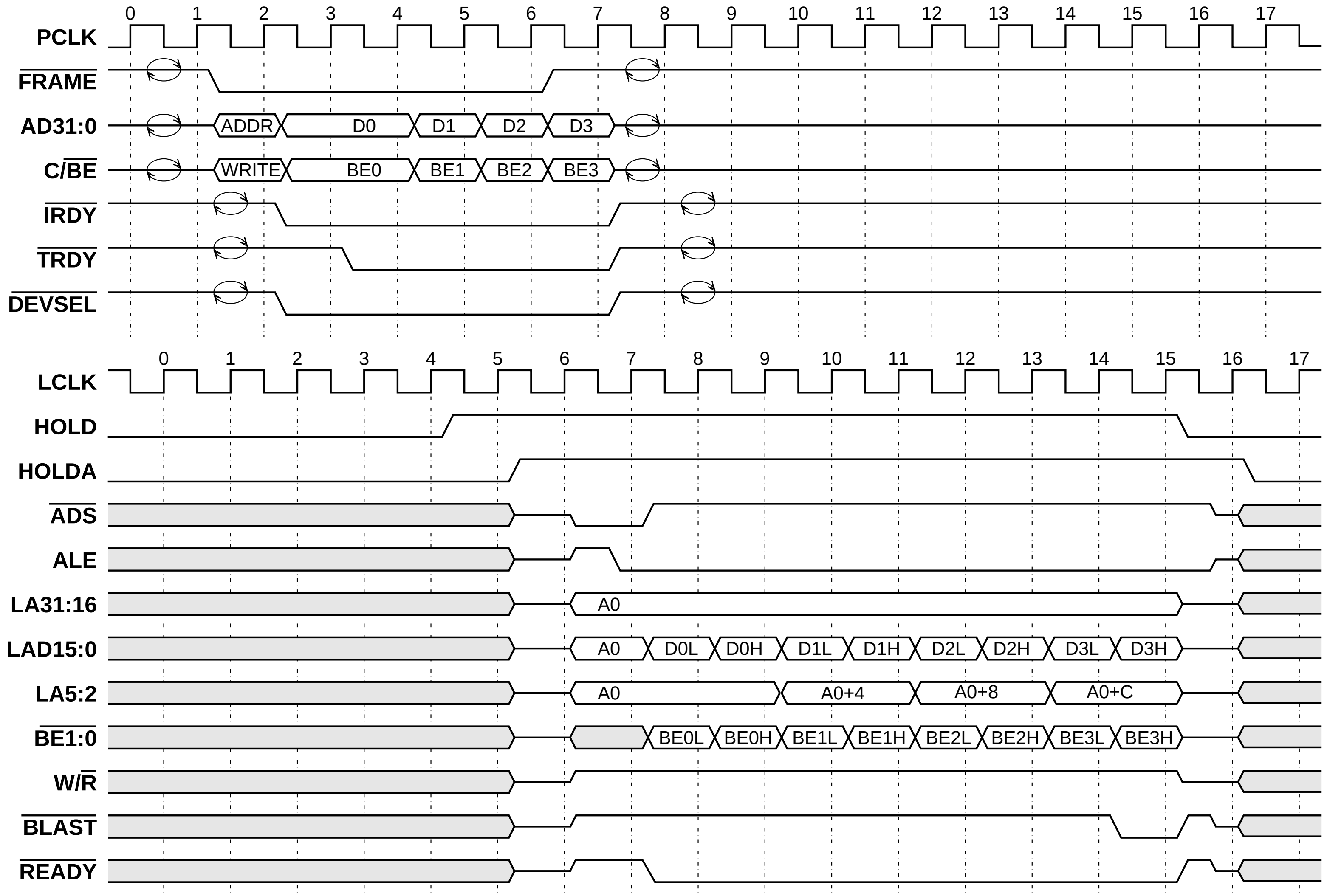


Figure 32: V350EPC (960 mode) Initiated Local Write Cycle

Figure 33: V350EPC (961 mode) Initiated Local Write Cycle

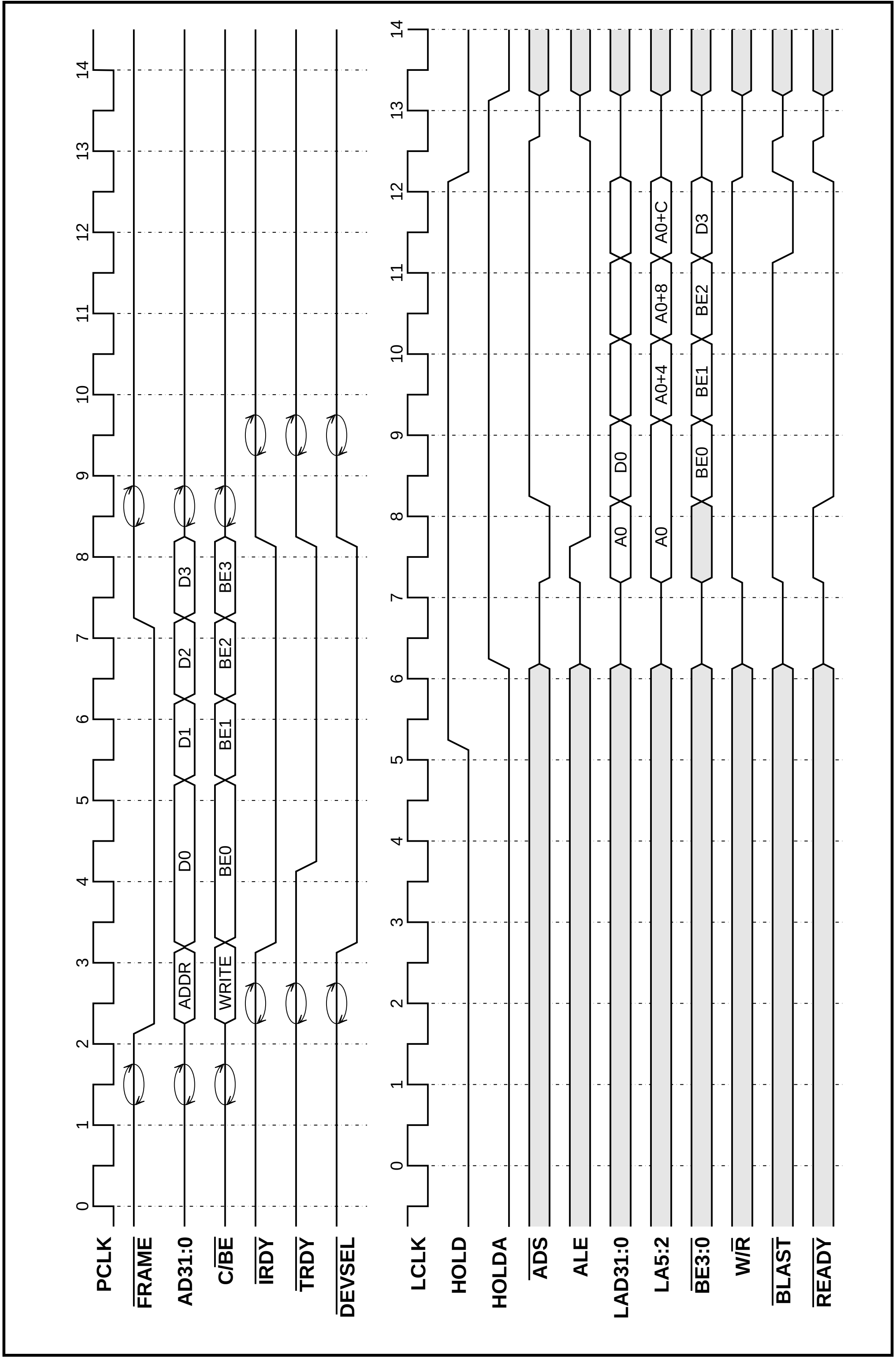


Figure 34: V360EPC (962 mode) Initiated Local Write Cycle

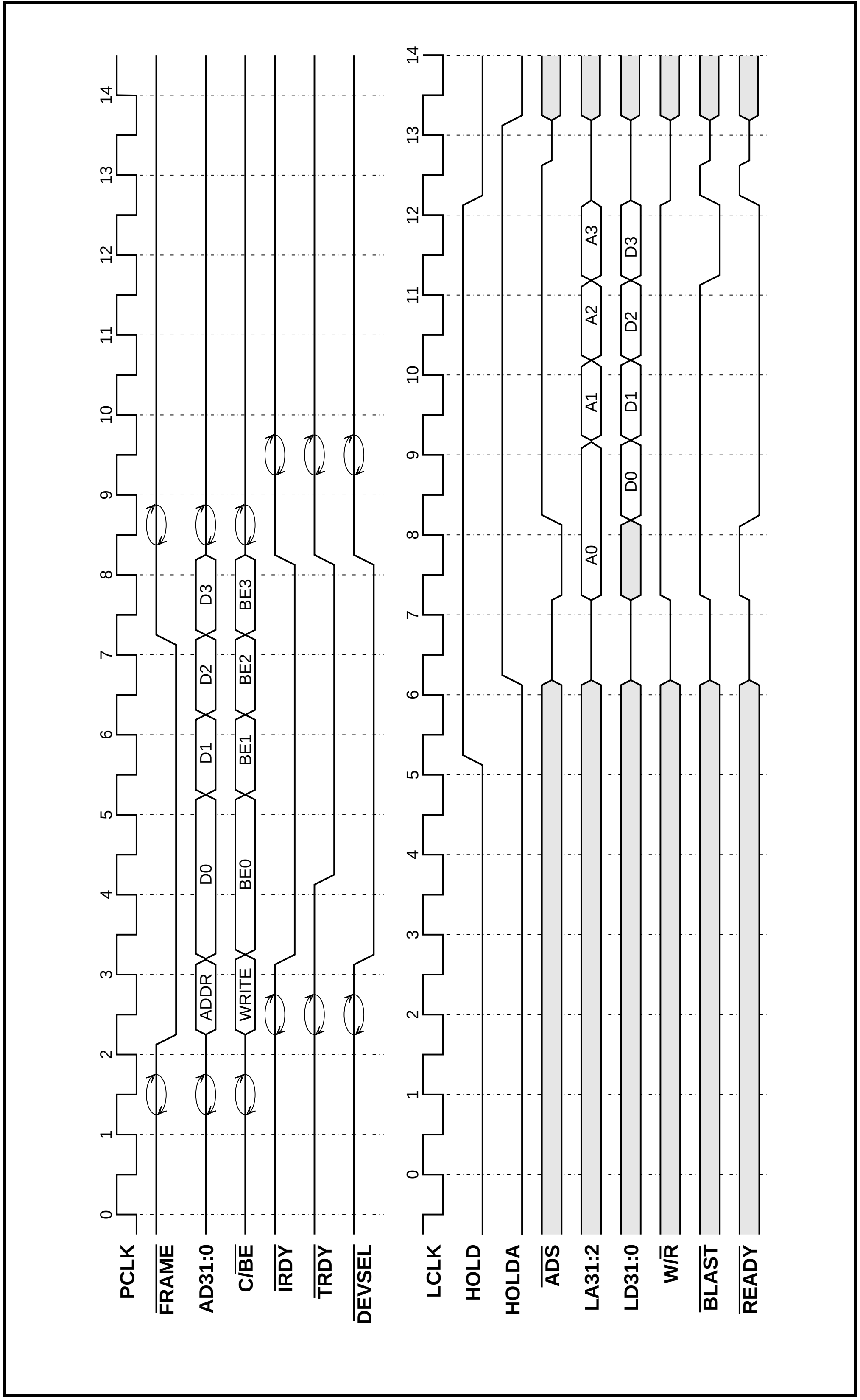
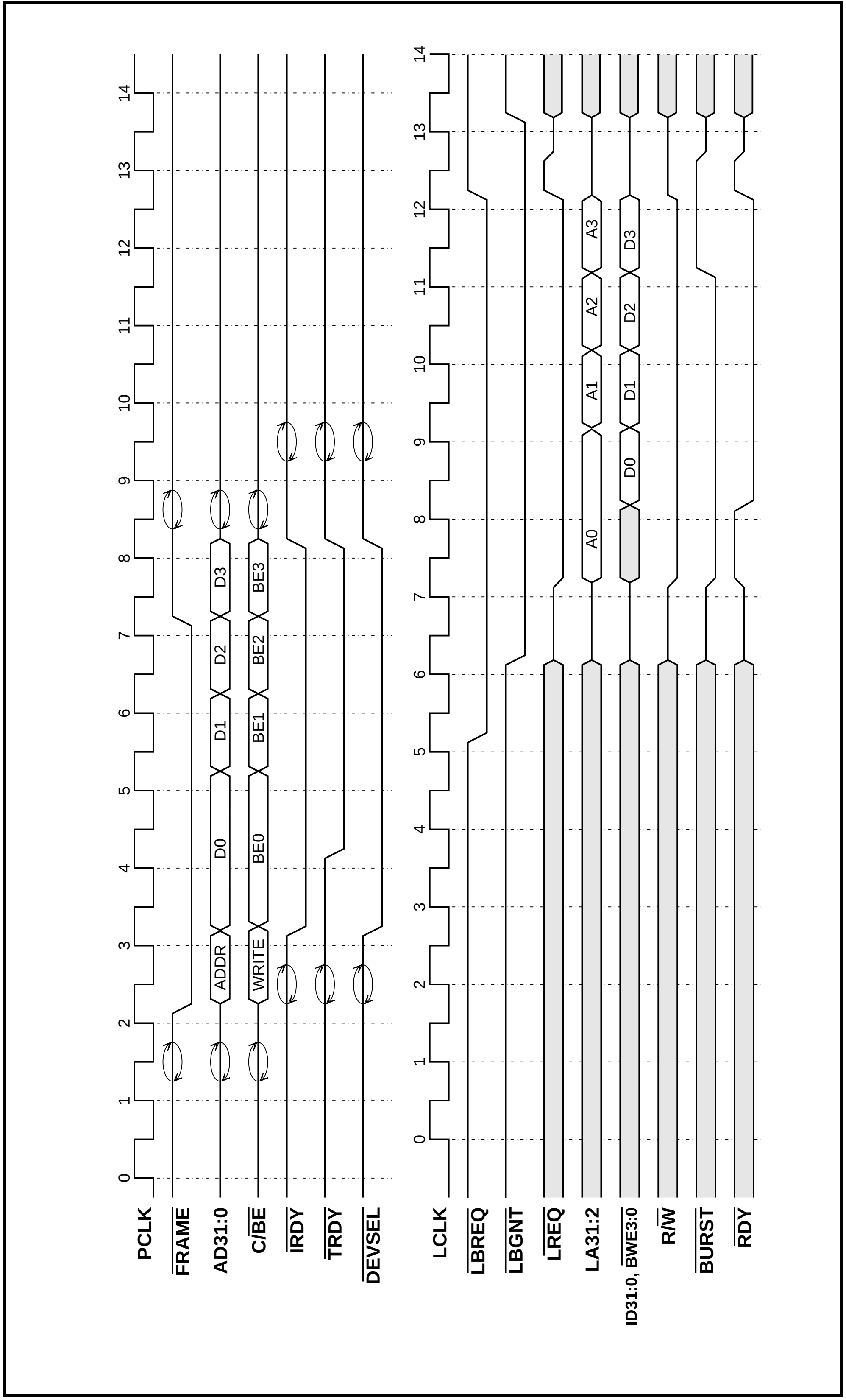


Figure 35: V360EPC (292 mode) Initiated Local Write Cycle



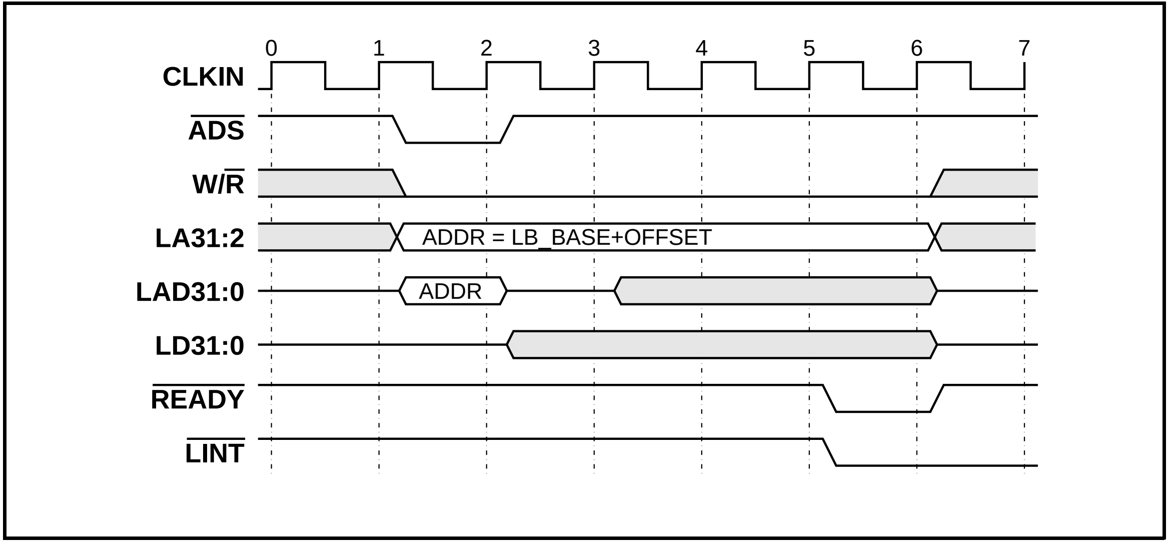
8.1.5 Target Mode PCI Error Signalling

Target mode reads and writes to/from PCI space may cause a number of PCI errors (see "PCI Interface"). The EPC reports these errors for reads by:

- Asserting $\overline{\text{RDY}}$ to unlock the local bus.
- Generating a maskable interrupt to both the local and PCI interrupt controllers.

Figure 36 shows the case of a PCI read error caused by a Master Abort (attempt to access a non-existent PCI device). Write errors are handled differently since they will complete on the PCI bus some time after the local master has posted them in the write FIFO. Write errors are reported only through the PCI error interrupt. See "PCI Bus Interface" for more details.

Figure 36: PCI Error Signalling (all bus types)



8.1.6 Deadlock Conditions and Resolution

There is a potential deadlock condition that exists whenever two or more local processor/ bridge combinations are used in a system. This condition manifests itself as follows:

- One local master attempts to read the local memory of another processor on the PCI bus; simultaneously the other processor is attempting to read the first processor's local memory.
- Per the PCI spec, the master in control of the PCI bus receives a "retry", however, since that master is performing a read it cannot return data to its processor, and therefore maintains a NOT READY indication to that processor. This situation effectively keeps the other processor involved off the local bus.

Since neither of these reads can proceed, the PCI bus is at a *deadlock*. If the write FIFO becomes full then there is also a similar deadlock issue involving writes.

The EPC resolves this problem automatically by detecting “overdue” unresolved reads or writes on the local bus. A timer can be enabled (LB_CFG register) that will time-out on any access that takes longer than 64-256 clocks to complete will be deemed overdue, and can be terminated by asserting RDY and/or LINT (to indicate a PCI access error). A software handler must be used to recover from the error. For processors that provide a bus backoff or retry mechanism (such as the i960Cx/Hx processors and the Am29030/40 processors) fully transparent deadlock avoidance is possible as described in section “Deadlock Avoidance using the BTERM as an Output” on page 81.

8.2 MASTER MODE

The local bus interface is said to be in *master mode* when the EPC is performing local read and write requests in response to PCI or DMA accesses.

8.2.1 Requesting the Local Bus

The local bus is requested by the EPC by asserting the HOLD(LBREQ) signal. Local bus access is granted by the local bus arbiter by returning the HOLDA(LBGRT) signal.

8.2.2 i960 Local Bus Reads and Writes

The local bus protocol used by the EPC family components is identical to that of the target processor: V960EPC duplicates the i960Sx protocol, the V961EPC duplicates the i960Jx protocol, and the V962EPC duplicates the i960Cx/Hx protocol.

All EPC’s support bursts longer in length than the i960 processor limit of 4 words (8 transaction on the i960Sx devices). The maximum burst length supported is 256 words. The end of the burst is indicated by the BLAST signal, as is the case with "standard" i960 buses. The burst length on the local bus is programmable via the LBRST_MAX field in the FIFO_CFG register. Bursts may also be terminated at any time by asserting the BTERM signal. The burst will re-start with another ADS assertion at the point it was terminated.

V3 has chosen not to duplicate the wait-state control logic found in the i960Cx (MCON registers) and i960Hx (PMCON registers).[1]

8.2.3 Am29K Local Bus Reads and Writes

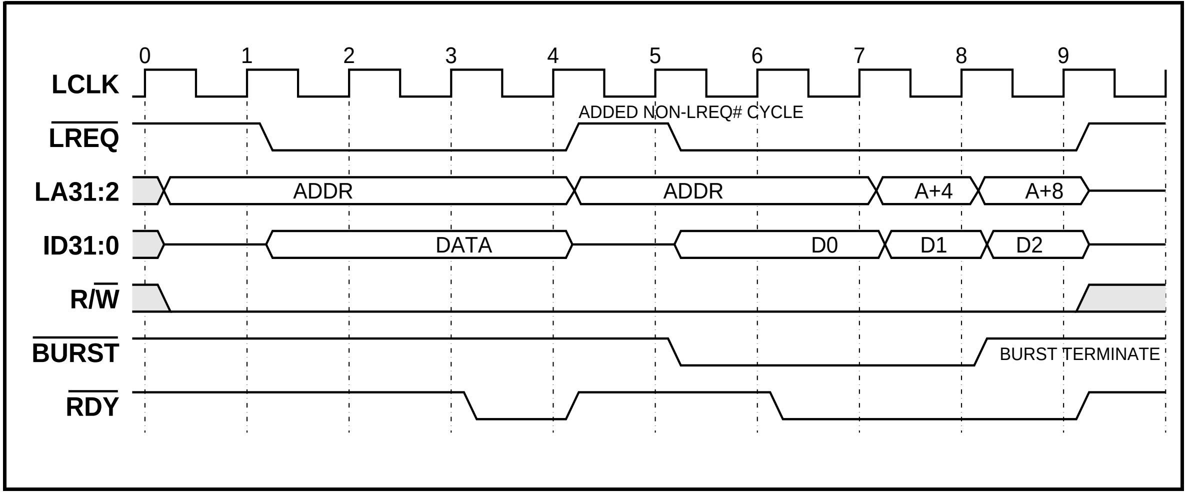
The V292EPC provides two different local bus protocols for Am29030/40 style busses: high-performance mode and strict compatibility mode. The two modes differ in the BURST signal assertion timing and in the length of bursts supported.

[1] This logic is of no use in systems using DRAM as main memory, as the wait-state profile for DRAMs is indeterminate (due to refresh cycles.)

8.2.3.1 Strict Compatibility Mode

In *strict compatibility* mode the BURST signal is asserted coincident with the assertion of LREQ (see Figure 37). This mode duplicates the bus protocol shown in the Am29030/40 documentation. When a series of access cycles are done by the V360EPC (292 mode) on the local bus (such as a series of burst writes) the LREQ signal will be de-asserted for one cycle between bursts. This is done so that the BURST and LREQ signals can occur simultaneously at the beginning of a burst or non-burst cycle.

Figure 37: V360EPC (292 mode) Local Bus Master Access (Strict Compatibility Mode)

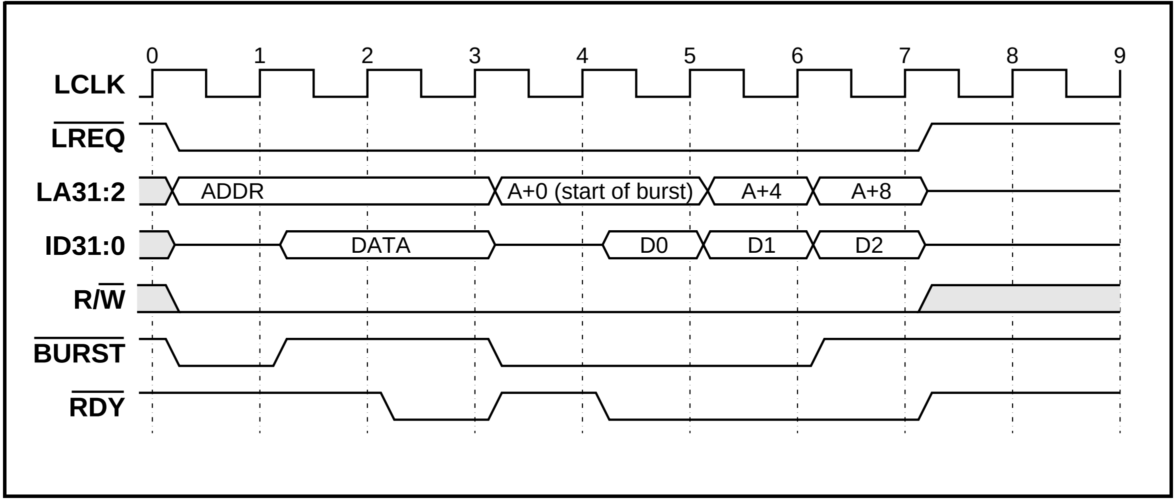


8.2.3.2 High-Performance Mode

In systems that choose to use high-performance mode, both BURST and LREQ are asserted as soon as the V360EPC (292 mode) begins a cycle (LCLK 1 and LCLK 3 in Figure 38). If the cycle is a non-burst cycle then BURST is de-asserted on the next cycle (LCLK2). If a burst is really going to happen then BURST remains asserted until before the last data transfer of the burst (LCLK 6) as it normally would. If the local bus slave devices don't care about the state of BURST on the first cycle then the high performance mode will save one cycle and should be used by setting the FAST_REQ bit in the SYSTEM register.

The V360EPC (292 mode) defaults to strict compatibility mode following reset. The local bus mode is changed by writing to the FAST_REQ bit in the SYSTEM register. System using the V292BMC/CMC Burst DRAM Controller can take advantage of the high performance mode and will also work when strict compatibility mode is selected.

Figure 38: V360EPC (292 mode) Local Bus Master Access (High-Performance Mode))



8.3 BURST SUPPORT

The EPC is designed to accommodate bursting of up to 1K bytes on both the PCI and local buses. Long bursting is accomplished by programming the PBRST_MAX and LBRST_MAX bits in the FIFO_CFG register to select the 1K byte (256 word) burst size. Simply selecting a 1K burst size will not guarantee a full 1K uninterrupted burst - other parameters must be considered that will limit the burst size:

- For aperture writes (local bus master writes PCI bus *or* PCI bus master writes local bus) the burst length on the source bus will determine the length on the destination bus if it is shorter than the value of the corresponding PBRST_MAX or LBRST_MAX.

- For the case of local to PCI access: burst length will be limited by PCI target devices by asserting STOP (bus retry) at its burst limit.

- In the case of read prefetch, bursts will terminate if the prefetch FIFO becomes full.

- Burst length on the local bus will be limited by the memory systems ability to support long bursts and not by the burst size of the processor on the local bus. Controllers such as the V96SSC, V96BMC, V96CMC, V292BMC and V292CMC will all support 1K bursts. Attempting to do long bursts to memory systems incapable of supporting them will result in lost or corrupt data.

- When using the V96BMC Rev D with the V350EPC or V360EPC please note that the BMC Rev D device strictly follows the i960 bus protocol which does not allow byte enables to change on a burst. However, the EPC has been allowed to be more flexible by allowing the byte enables to change in a burst. Therfore, you need to take this fact in consideration when you are using the V96BMC Rev D with the EPC.

```
---------------------------
---------------------------
-- V3 Modification (VHDL CODE) for a Small PLD:
-- Explanation:  BTERM# must be driven only when a write
--               cycle is being initiated by the EPC when not all
--               the byte enables are driven.
--
--               This fix creates a better cohesion between the BMC
--               and the EPC since the BMC adheres strictly
--               to the i960 protocol and the EPC has more flexibility
--               to allow byte transfers to occur. Please note that
--               this fix is not required when an SSC is used as the
--               memory controller. (This occured because the
--               BMC was designed before the EPC was designed,
--               and the SSC came after the EPC).
--

process
begin
wait until(clkin'event and (clkin = '1'));

-- identify the troublesome cycle by EPC
fix_qads <= (not l_ads_n) and pci_hlda and l_wr_n;

end process;

-- disconnect (block) the troublesome cycle from EPC
l_bterm_n = '0' when( not((l_be_n3 & l_be_n(1 downto 0))
                  = "000") and fix_qads='1') else
                 '1' when(pci_hlda = '1') else 'Z';

---------------------------
---------------------------
```

Bursts by the EPC are always terminated at a burst modulo page boundary. For example: if a 1K burst length is selected then the EPC will never cross a 1K page boundary (it will never burst from address XXXXX3FCH to XXXXX400H). Similarly, if a 16 byte (4 word) burst is selected it will never burst from XXXXXX0CH to XXXXXX10H. This makes the EPC comparable to processors and page mode memory devices.

8.4 BTERM OPERATION (961 AND 962 MODE ONLY)

This section is applicable only to the EPC in 961 and 962 modes.

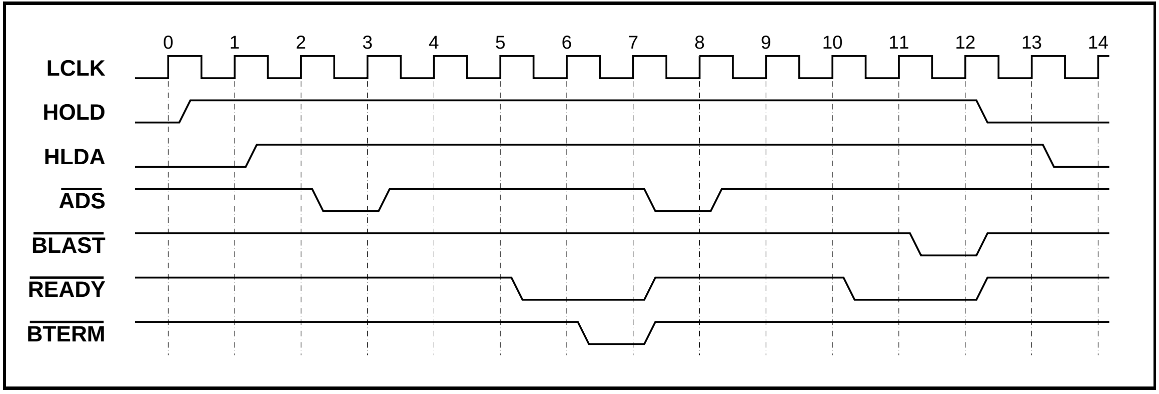
The BTERM signal is used in two ways. When the EPC is a local bus master, the BTERM input acts as a "Burst Termination" input with the same timing as the READY input. When the EPC is a local bus slave for local aperture to PCI read/write operations, the BTERM signal can be programmed to drive as an output for time-out conditions.

Note that BTERM must have a pull up resistor on the pin even if it is not being used in as an input or output as described below.

8.4.1 BTERM as an Input

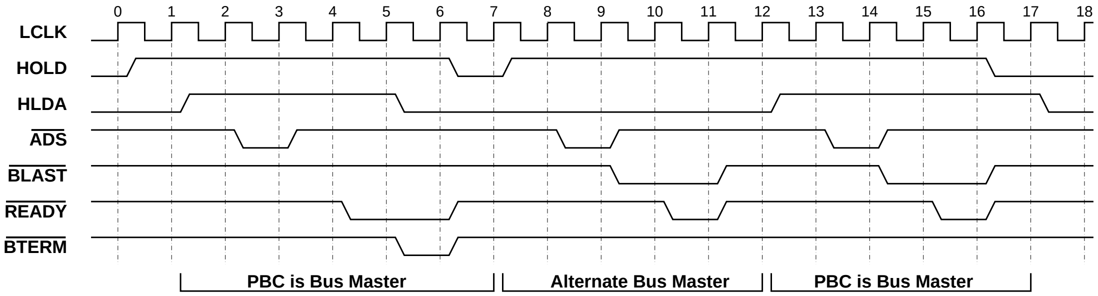
The EPC will respond to BTERM together with READY as if BLAST had been asserted also with READY. This will cause the ADS signal to be re-driven to start a new cycle where the original burst left off. The sequence is depicted below: It begins as a normal EPC local bus master cycle with HOLD/HLDA/ADS being driven in sequence. At LCLK 6 the first data is READY and the cycle looks like a normal burst. However, at LCLK 7 BTERM is asserted with READY and the cycle terminates as if BLAST had been asserted at that time. At LCLK 7 the EPC begins to drive ADS again and with the address for data item 3 left over from the previous unfinished burst. This burst cycle terminates normally with BLAST and READY at LCLK 12.

Figure 39: BTERM as in Input



Additionally, the assertion of BTERM can be used in conjunction with the de-assertion of HLDA to give another local bus master ownership of the bus so that the EPC won’t "hog" the bus when long bursts are performed. When a burst is terminated (with or without BTERM asserted) and more data transfer is pending, the EPC will drop its’ HOLD request for one cycle at the end of the burst if the HLDA signal is taken away before the end of the cycle thus allowing another higher priority master to gain control of the bus. This is NOT the way that i960 processor usually drives its’ HLDA output. Thus HLDA back to the EPC must be driven by a gated version of HLDA from the processor - typically from an arbiter device. The following diagram (Figure 40) illustrates the sequence.

Figure 40: Early Suspension of EPC Local Bus Ownership



In summary, the BTERM used as an input will terminate bursts similar to a PCI burst disconnect. When the HLDA input is used with BTERM, a high priority local bus device can gain access to the local bus by kicking off the EPC early.

8.4.2 Deadlock Avoidance using the BTERM as an Output

The BTERM signal can be used to indicate a time-out condition when a local bus master is trying to access the PCI bus through one of the EPC apertures. This operation is enabled through the LB_CFG register bit 1. When enabled, a time-out condition (as defined in the description given in the LB_CFG register) will cause the normally high-Z BTERM signal to be driven low for one LCLK and then high for one clock before being returned to its' high-Z state. A time-out condition is usually caused by a deadlock situation, where the local processor is attempting to access resources on another similar subsystem while the local processor of that other subsystem is attempting to access resources on the local bus of the first processor. On processors (such as the i960CA) that have a BOFF input, it is possible to break the deadlock by generating a BOFF from the BTERM output of the EPC. This is accomplished with a simple modification to the local bus arbiter device (typically a Programmable Logic Device). An example implementation is given below.

Figure 41: Circuit for Generation of Processor BOFF from V962EPC BTERM Output

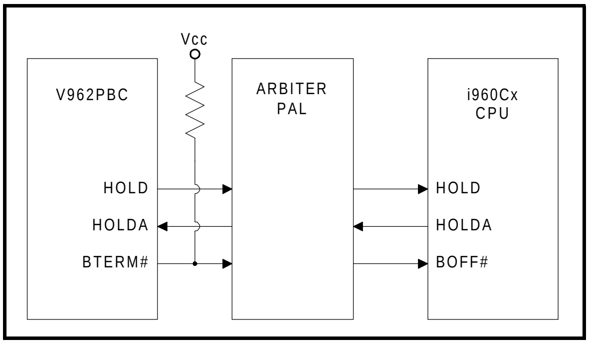
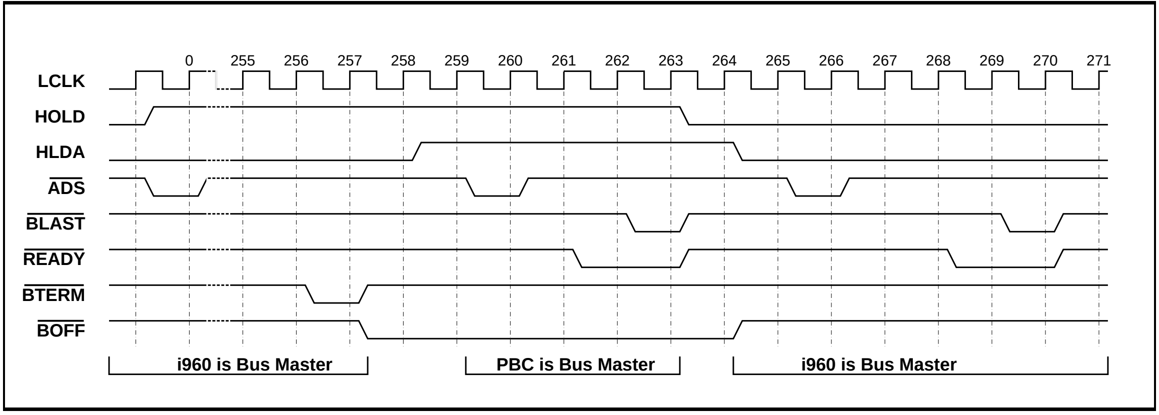


Figure 42: Waveform Showing Deadlock Avoidance Using BTERM and BOFF



BTERM at LCLK 257 is detected by the Arbiter and used to generate BOFF. When BOFF is asserted to the processor, it will cause it to float most of its signals as if it had lost mastership of the bus. Internally the processor is held as if in wait state waiting to see a READY assertion. This gives the EPC a chance to take mastership of the bus when it receives HLDA (hold acknowledge). With its' data transfer complete at LCLK 263, the EPC de-asserts its hold request. BOFF is then de-asserted to the processor causing it to reassert its' ADS strobe. This restarts the previously timed out access via the EPC to PCI address space. With the deadlock broken, this access now completes normally. This provides an automatic hardware solution to the deadlock problem: no deadlock handling software is required.

8.5 LOCAL BUS PARITY

The EPC is capable of generating and checking data parity on the local bus. When acting as a local bus master, the EPC generates one bit of parity information for each byte of data during write cycles. During read cycles by the EPC or any other local bus master, the EPC can perform parity checking.

Parity generation and checking is intended for operation with an external memory system or peripheral device. It is commonly used with DRAM arrays where single bit errors can occur due to the nature of these devices. Consequently, valid parity is not asserted when the EPC is a slave device on the local bus. In this case the value of the parity signals should be treated as "unknown".

8.5.1 Relationship between Local Parity and PCI Parity

There is no direct relationship between PCI bus parity and local bus parity since they both operate in a different manner and for a different purpose. PCI parity is provided as a means to ensure the integrity of point-to-point, master-to-slave connections. Local bus parity, on the other hand, is designed to check the integrity of a local bus memory system typically implemented with DRAM. This requires that the local bus parity information travel on the bus with the same timing as the data. This is not the case with PCI parity which lags the data by one clock cycle.

.

Table 11: Comparison of PCI and Local Parity

	PCI Parity	Local Parity
Number of parity signals	1	4
Parity calculation	1 bit for: AD[31:0], C/$\overline{\text{BE}[3:0]}$	1 parity bit per data byte
Parity generation	generated by receiver of data	generated by bus master
Data to parity relationship	parity lags data one clock	parity and data together
Data to parity error relationship	error lags data two clocks	error lags data one clock

8.5.2 Local Bus Parity Generation

Valid byte parity is driven out on to the LPARx pins whenever a master write cycle is performed and with the same timing as write data is driven onto the data bus (or LAD bus in the case of the V350EPC). This allows the local bus memory system to store the parity information as if it is extra data. This differs from PCI parity in that PCI parity lags the data by one clock cycle making it more difficult to store along with the data in a memory array (this is not the intention of PCI parity).

The EPC generates parity ONLY for itself when acting as a local master (PCI-to-local aperture writes and PCI-to-local DMA transfers). Other masters (such as the CPU) must generate their own parity when they perform write cycles. Parity is also NOT generated by the EPC when it is a local bus slave (either for access to the internal registers or for Local-to-PCI aperture access).

The four parity bits are generated according to Table 12:

Table 12: Relationship between Parity Output Signals and Output Data

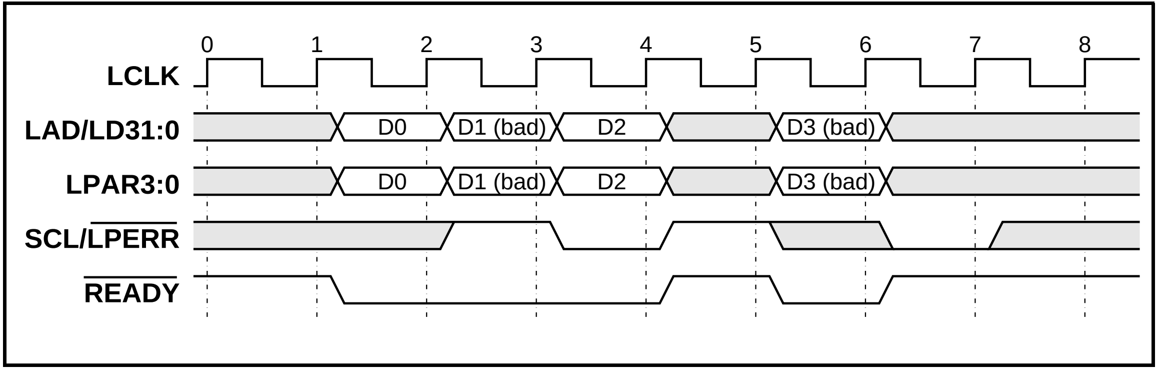
LPAR3	xor(LD[31:24], POE[a])
LPAR2	xor(LD[23:16], POE)
LPAR1	xor(LD[15:8], POE)
LPAR0	xor(LD[7:0], POE)

[a] POE bit of the SYSTEM register (Parity Odd/Even)

8.5.3 Local Bus Parity Checking

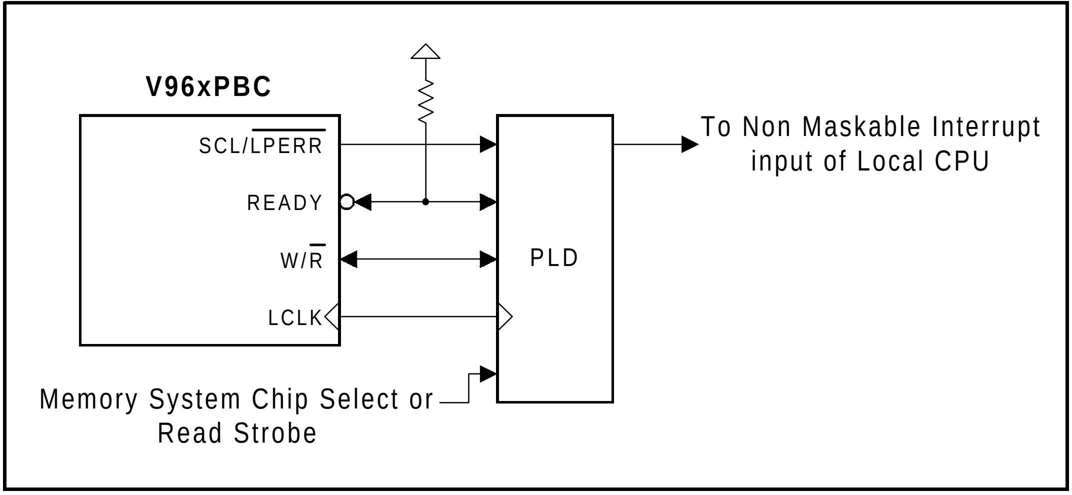
Each LCLK cycle, the EPC checks parity on the local bus and drives the SCL/$\overline{\text{LPERR}}$ signal accordingly on the next cycle. Consequently, the SCL/$\overline{\text{LPERR}}$ signal must be qualified to allow only valid regions of address space to be checked. Figure 43 shows the relationship between data, parity and the parity error output. Note that the parity error output $\overline{\text{LPERR}}$ lags the parity and data by one clock.

Figure 43: Timing Relationship Between Parity, Data and the Parity Error Output



The circuit in Figure 44 is an example of how the $\overline{\text{LPERR}}$ signal should be qualified to generate a high priority interrupt to signal a parity error to the local processor.

Figure 44: Qualification of Parity Error to Generate High Priority Interrupt


V96xPBC
SCL/$\overline{LPERR}$
READY
W/$\overline{R}$
LCLK
PLD
To Non Maskable Interrupt input of Local CPU
Memory System Chip Select or Read Strobe

PLD equations

```
qual.d = ready & !wnr & mem_cs;/* intermediate qualifier signal */

nmi.d  = lperr & qual /* Non Maskable Interrupt to CPU
```

Chapter 9 PCI Configuration

The EPC may be used as both a host or target bridge device. As such, the EPC can both generate configuration cycles and respond to them. This chapter describes both types of PCI configuration.

9.1 CONFIGURATION AS A SYSTEM HOST BRIDGE

The EPC acts as a host bridge when it is used to configure other PCI devices in the system. For example, a laser printer that uses an i960® as the main CPU and uses PCI as the mezzanine bus, would run configuration cycles to initialize PCI peripherals such as ethernet chips and SCSI controllers.

9.1.1 EPC Host Configuration Mechanism

PCI configuration cycles consist of setting a specified target's IDSEL line active, then performing reads and writes to the configuration space of the selected peripheral. The IDSEL line is deasserted after accesses to the target peripheral's configuration space are complete.

9.1.2 Controlling Target IDSEL Lines

The EPC does not provide a direct control of the state of individual targets' IDSEL lines. The system hardware must provide a mechanism for activating the IDSEL line for each target (if the EPC is to be used as a host bridge). IDSEL is not like other PCI signals in that it need not be synchronous with each configuration cycle. IDSEL must be active during the address phase to select the EPC.

An external register is the simplest method for controlling IDSEL. V3's Am29K™ PCI motherboard (the Lion-29K), for example, uses a PAL device that sets and clears specific IDSEL lines based on accesses to specific memory locations in local space.

9.1.3 Generating Configuration Reads and Writes

The address placed on the bus during configuration cycles is actually encoded information as shown in Figure 45. The EPC does not automatically generate this encoding, it is completely under software control. The configuration address encoding is programmed as follows:

- Setup one of the Local-to-PCI apertures as 16 megabytes in length. This forces the address translation logic to pass the lower 24 bits of the address through to the PCI bus *without* translation. The lower 24 bits of the address are the encoded configuration information. The base address in Local memory for this aperture may be placed on any 16Mbyte boundary.

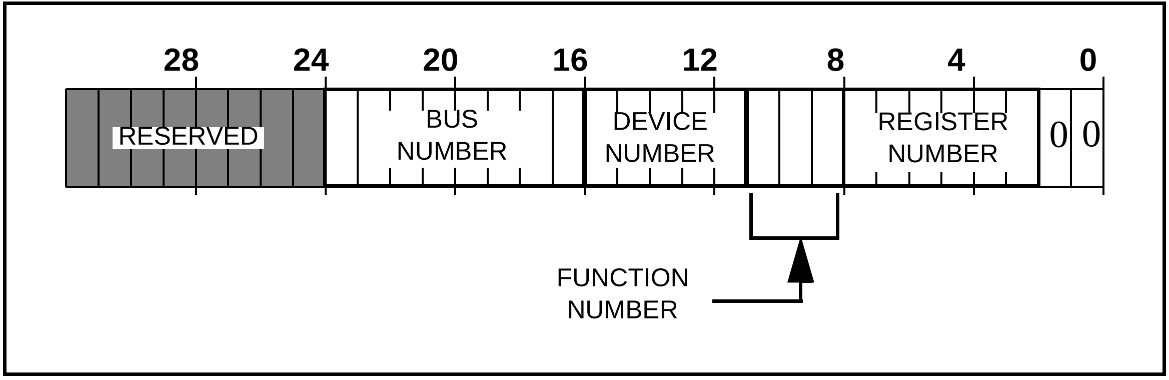
- Setup the address translation for the above aperture to translate accesses to 0000.0000H. Since the aperture is setup for 16Mbytes, this has the effect of setting A[31:24] to zeros. Set the TYPE field for the Local-to-PCI aperture to “Configuration Reads/Writes”.

- Create a 24-bit value that encodes the configuration address information. Add the base address of the Local-to-PCI aperture to this encoded information.

- Perform a read (or write) to the above address in local space. This will be translated to a configuration read (or write) in PCI space.

In the above example, it is assumed that the IDSEL lines are set appropriately by the user’s system hardware and software.

Figure 45: Encoded Configuration Address Information



9.1.4 Using Configuration Information

The following information is usually determined during configuration:

- What PCI devices are present in the system and what type of devices they are

- What resources are necessary for each device in terms of memory, I/O, and interrupts

- What capabilities each of the devices has (i.e. Fast Back-to-Back capable)

The PCI Specification describes in detail the information available during PCI configuration and how to retrieve that information. The usage of that information is highly application dependent and is beyond the scope of this manual.

9.1.5 Determining the Presence of Target Devices During Configuration

A Master Abort will occur if the system host attempts to access the configuration space of a device that is not present. This occurs, for example, when the IDSEL line for an empty PCI slot is activated and a configuration cycle is run. When this occurs, the EPC sets the Master Abort bit in the PCI_STATUS register and returns FFFF.FFFFH to the host. This combination of events is used to determine the presence of PCI devices in a system following a reset.

Typically, the following mechanism is used during the boot code:

- Each IDSEL line is activated in turn and a configuration read is attempted from the Vendor ID register

- If a vendor ID of FFFFH is detected it means that the current IDSEL line is not connected to a valid device (FFFFH is a reserved Vendor ID for just this purpose)

- If a valid vendor ID (i.e. not FFFFH) is detected, further configuration of the device is performed

Most systems will elect to disable PCI error interrupts during configuration.

9.2 CONFIGURATION AS A TARGET BRIDGE

The EPC responds to type 0 configuration cycles when acting as a target bridge. Type 1 (PCI-to-PCI bridge) configuration cycles are ignored.

The following information is usually retrieved from a target device during configuration:

- The number and size of I/O and memory address regions required (read from the PCI base registers)

- The vendor ID, device ID, and device type information from the PCI header

- Any supplemental information provided for in the PCI header

9.2.1 EPC Base Register Response to Configuration Inquiries

Most information is read by the system host directly from the EPC’s configuration space registers. For example, Vendor ID is retrieved simply by reading the Vendor ID register.

The following are exceptions to the above:

- Size requested for PCI-to-Local data transfer apertures (PCI_BASEx registers).
- Size requested for expansion ROM transfer apertures (PCI_ROM register).

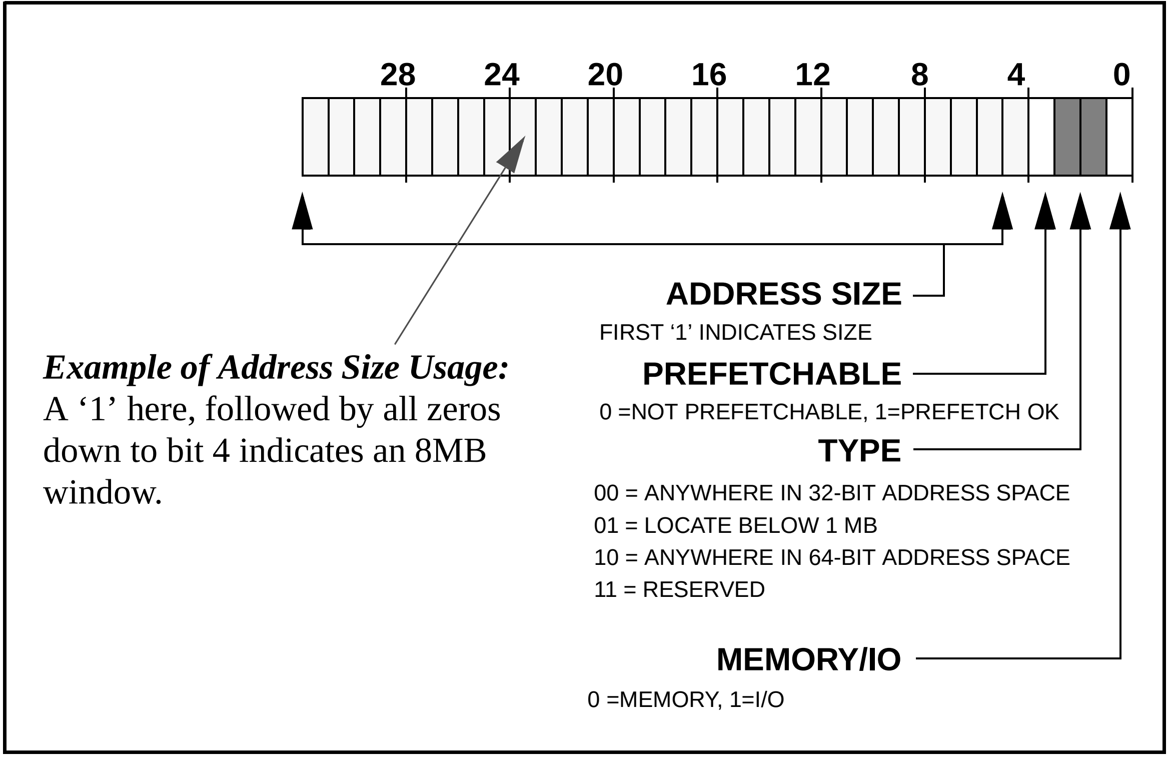
In both of these cases the PCI Specification outlines a two step interrogation process. First, each base address register is written with FFFF.FFFFH. Then the base register is read back. The value read back indicates the size and capabilities of the aperture (see Figure 46).

As an example, let's assume that PCI-to-Local bus aperture 0 is programmed for 4 megabytes in memory space with prefetching enabled. When the system host interrogates this register it will read back FFC0.0004H which is interpreted as:

- The aperture is four megabytes in size, since the first "1" seen in the address field (scanning up from bit 4) is at the 4M level
- The aperture can be located anywhere in the 32-bit address space on a 4MB boundary
- The aperture is memory mapped
- The aperture is "prefetchable"

The EPC sets the size information according to the size of the aperture specified in the PCI_MAPx register. Unused base registers in the EPC return all zeros.

Figure 46: Base Register Return Information


28
24
20
16
12
8
4
0
ADDRESS SIZE
FIRST '1' INDICATES SIZE
PREFETCHABLE
0 =NOT PREFETCHABLE, 1=PREFETCH OK
TYPE
00 = ANYWHERE IN 32-BIT ADDRESS SPACE
01 = LOCATE BELOW 1 MB
10 = ANYWHERE IN 64-BIT ADDRESS SPACE
11 = RESERVED
MEMORY/IO
0 =MEMORY, 1=I/O
Example of Address Size Usage:
A '1' here, followed by all zeros down to bit 4 indicates an 8MB window.

9.2.2 EPC Expansion ROM Base Register Response to Configuration Inquiries

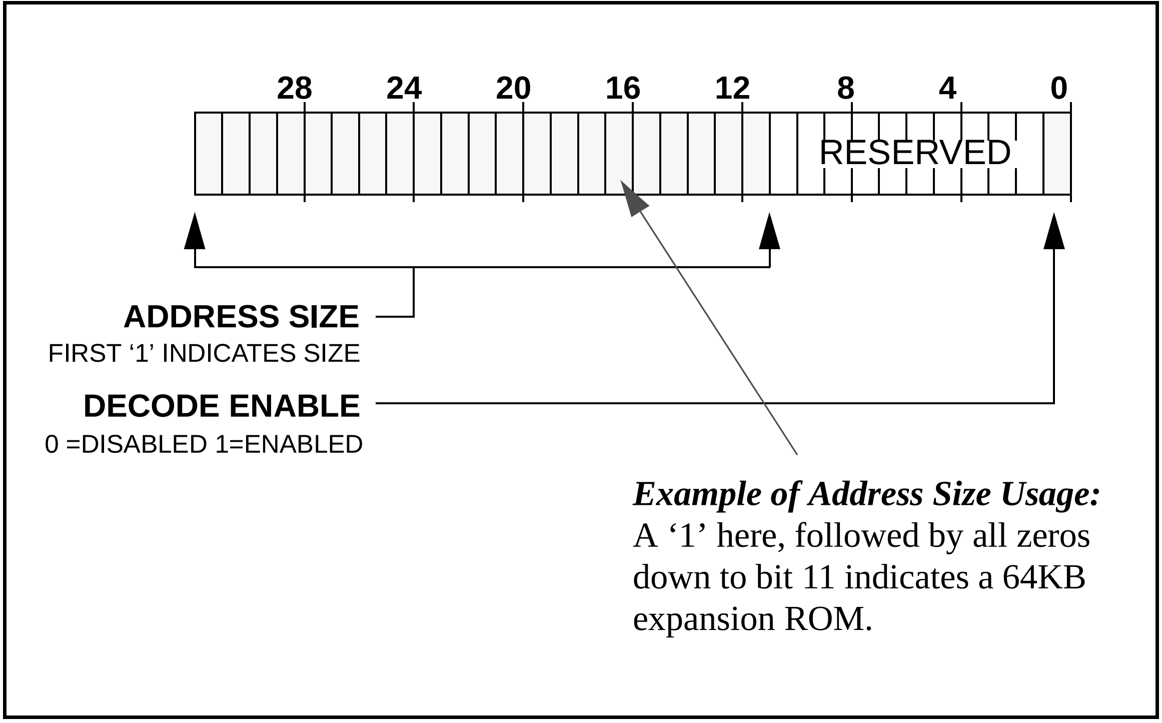
The expansion ROM base register is interrogated using the same method outlined above:

- All ones are written to the register (with the exception of the enable bit).
- The requested size is read back from the same register.

The format for the expansion ROM base return information is shown in Figure 47. The size information is determined from the expansion ROM size field set in the EPROM_MAP register.

Additional inquiries may be necessary depending on the host platform architecture. PCI based PCs, for example, will map the expansion ROM into memory and then scan for the ROM present signature of 55AAH. Please refer to the PCI BIOS spec for your particular target architecture for more information.

Figure 47: Expansion ROM Register Return Information


28
24
20
16
12
8
4
0
RESERVED
ADDRESS SIZE
FIRST '1' INDICATES SIZE
DECODE ENABLE
0 =DISABLED 1=ENABLED
Example of Address Size Usage:
A '1' here, followed by all zeros down to bit 11 indicates a 64KB expansion ROM.

Chapter 10 PC Compatibility

The EPC includes several features to improve compatibility when used as a target bridge in real mode PC/DOS systems. Real mode DOS systems have several limitations:

- Memory accesses are limited to below 1 megabyte. Add-in cards designed for real mode DOS typically use small blocks of memory located between A0000H and F0000H

- I/O devices use small blocks of I/O space, often with pre-defined addresses

The EPC provides a real mode DOS aperture than can be used to emulate DOS memory and I/O apertures for backward compatibility. It is important to note that these features fall outside of the PCI specification and can be disabled for strict compliance.

10.1 REAL MODE DOS COMPATIBILITY APERTURE

When the ADR_SIZE field of the PCI_MAP1 register (not available for Aperture 0) is configured for DOS compatibility mode then the PCI_BASE1 register is interpreted differently than the standard PCI definition. The need for this is a consequence of the fact that the original ISA bus based PC only decoded A9 down to A0 for I/O space address mapping. DOS mode addressing allows up to 3 holes in the PCI address space to be decoded that are useful for emulating DOS peripherals. There are 2 options depending on how the IO bit is set in the corresponding PCI_BASE register:

- IO = ‘0’ allows for up to 2 decoders for PCI I/O cycles in addition to a single memory decoder for PCI memory cycles in the region between 512KB and 1MB

- IO = ‘1’ allows for up to 3 decoders for PCI I/O cycles

Whenever a PCI cycle is detected by the DOS mode decoder (I/O or memory) then a corresponding bus cycle will be seen in a 1MB aperture on the local bus. The base address of that aperture is determined by the MAP_ADR field of the PCI_MAPx register. Within the local bus aperture, the first 64K will typically be used as the local bus remapping of PCI I/O cycles. For I/O cycles all relevant I/O addresses (up to A15) are mirrored on the local bus so that address aliasing information is carried across as DOS mode peripherals expect.

DOS memory region decode doesn’t remap the address space within the 1MB aperture. Therefore, if a region is decoded starting at PCI address 0xA0000 then it will appear at

0xA0000 + the 1MB region selected by MAP_ADR in the PCI_MAP register.

The operation of DOS mode decode can be more easily understood with the following diagrams.

Figure 48: DOS Compatibility Mode Memory Decoding and Address Translation

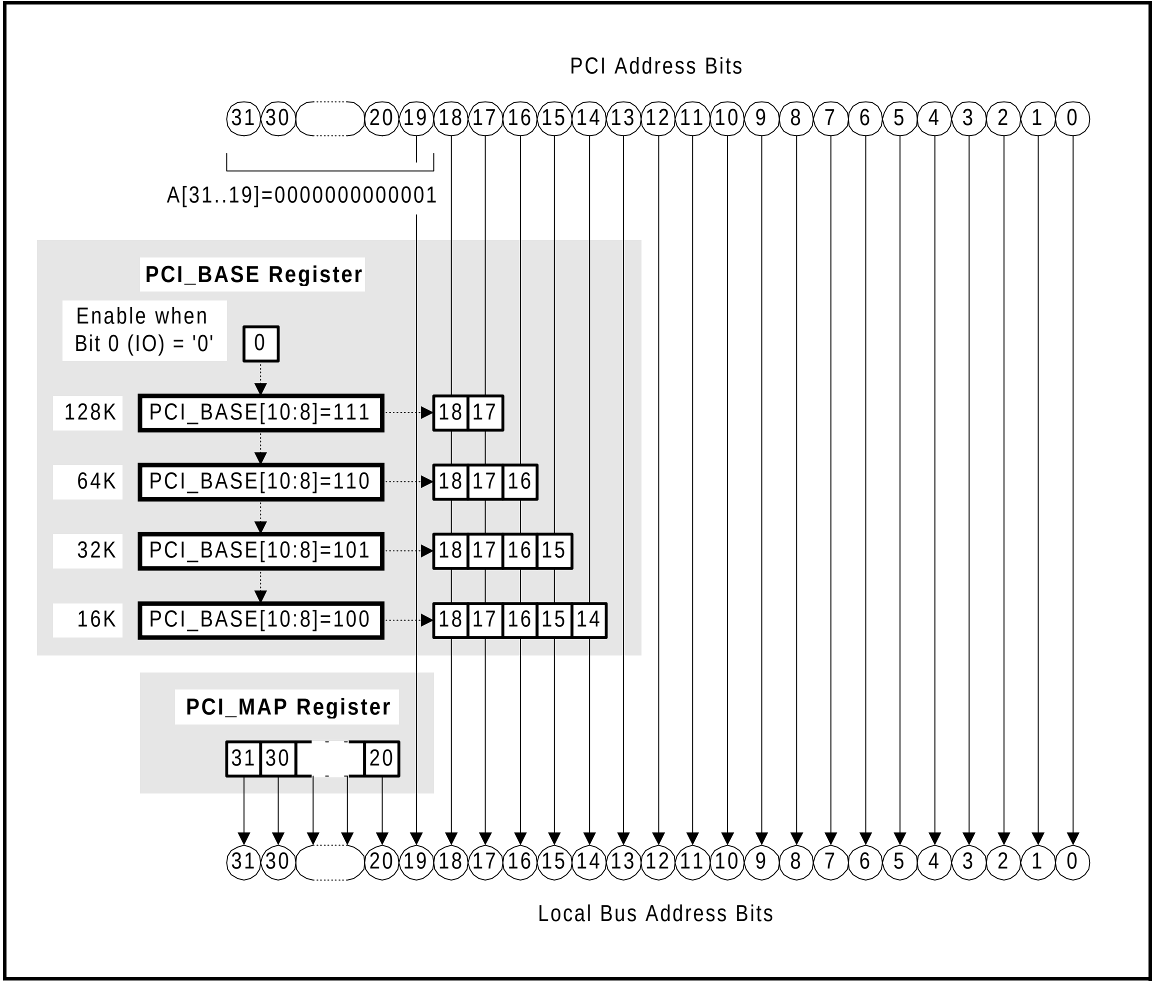
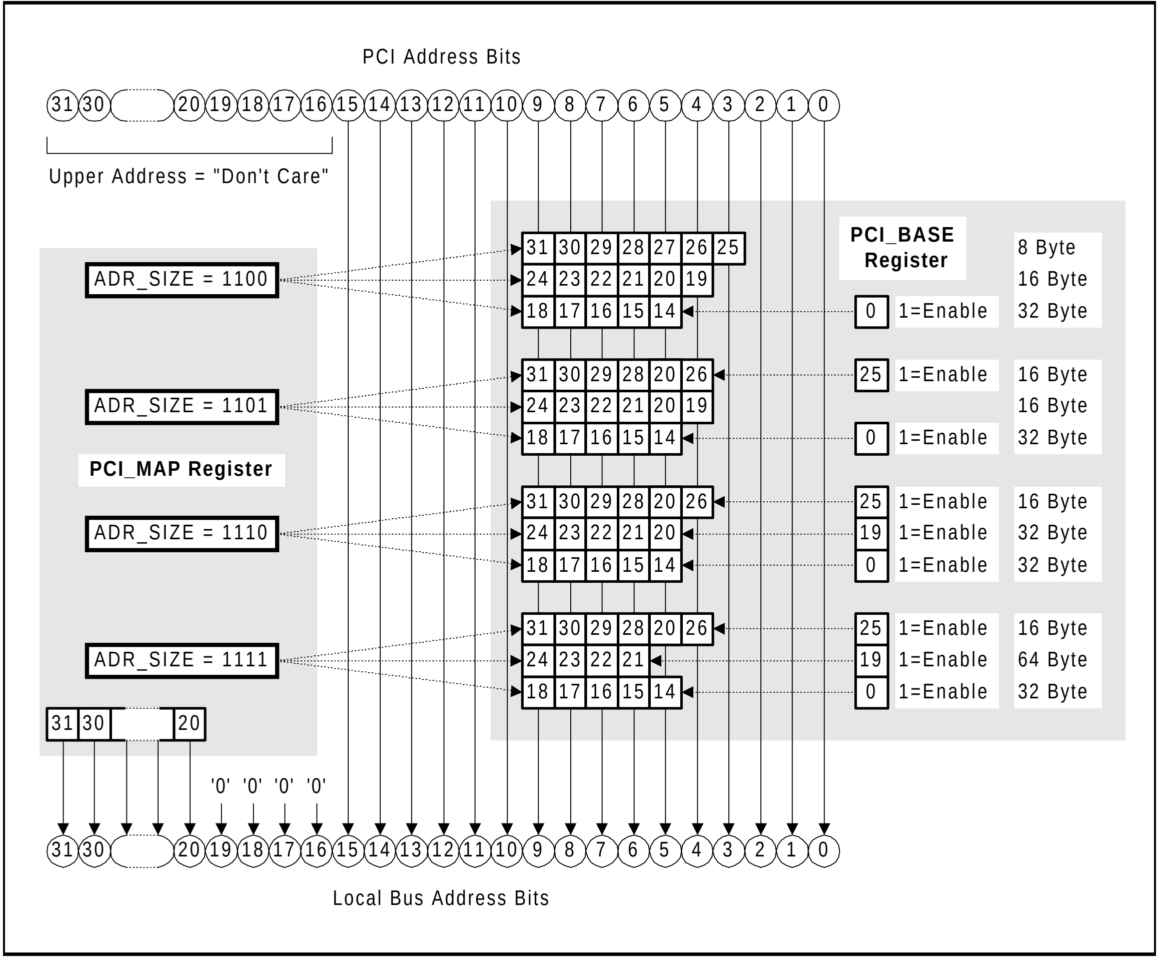


Figure 49: DOS Compatibility Mode I/O Decoding and Address Translation



10.2 EXAMPLE: VGA PERIPHERAL

Standard VGA addressing requires 3 address decodes:

- VGA Memory accesses - 0xA0000 - 0xBFFFF (128K bytes)
- VGA I/O accesses - AD[9:0] = 0x3B0 - 0x3BF (16 bytes) in addition to all ISA aliases where AD[15:10] are not decoded
- VGA I/O accesses - AD[9:0] = 0x3C0 - 0x3DF (32 bytes) in addition to all ISA aliases where AD[15:10] are not decoded

Such an address map can be decoded through a single aperture by programming one of the sets of PCI_BASE and PCI_MAP registers in the following way:

- PCI_MAP: ADR_SIZE = 1110 to select a 16 and 32 byte I/O decode.

- PCI_BASE[10:8] = 111 to select a 128K memory space.
- PCI_BASE: IO bit = 0 to enable the 3rd decode to be in memory space instead of I/O space.
- PCI_BASE[31:25] = 1110111 to enable a decode of AD[9:0] = 0x3B0 - 0x3BF for PCI I/O cycles.
- PCI_BASE[24:19] = 111101 to enable a decode of AD[9:0] = 0x3C0 - 0x3DF for PCI I/O cycles.
- PCI_BASE[18:17] = 01 to decode PCI memory cycles in the address range 0xA0000 - 0xBFFFF.

Chapter 11 Mailbox Registers

Occasionally it is necessary to pass small amounts of data or commands between the PCI bus and the Local bus. For example, an intelligent EPC based PCI disk controller may need to receive “get sector” commands from the x86 system host. Such commands could be sent to the local CPU by PCI-to-Local memory transfers into a command buffer, followed by a PCI interrupt request to indicate transfer completion. Using this method, however, is very inefficient, especially when only a handful of bytes need to be transferred.

To solve this problem, the EPC provides 16 mailbox registers which may be used to transmit and receive small amounts of data between the local CPU and the PCI bus. In addition, each mailbox register can request an interrupt to signal the receipt, or the demand, for more data.

Mailbox registers are also commonly used to emulate hardware registers in systems requiring backward register compatibility.

11.1 OVERVIEW

The EPC provides 16 8-bit mailbox registers arranged as contiguous bytes in both the Local and PCI internal register apertures. Each mailbox register is a dual ported memory, capable of generating an interrupt request whenever it is read or written from either side. Figure 50 shows a block diagram of the mailbox registers.

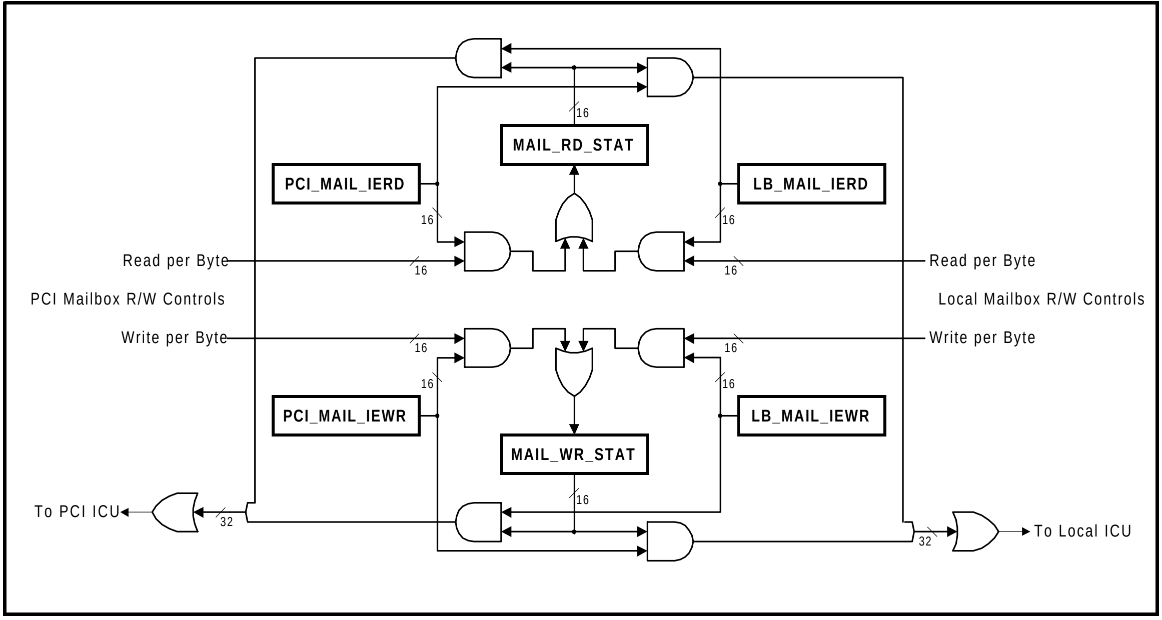
11.1.1 Accessing the Mailbox Registers

The mailbox registers are accessed from the Local bus through the Local-to-Internal Register (LB_IO_BASE) aperture. Typically, LB_IO_BASE is programmed during system initialization (see “Initialization”).

The mailbox registers are accessed from the PCI bus through the PCI-to-Internal Register (PCI_IO_BASE) aperture. Alternatively, the mailbox registers can be accessed through PCI configuration space for the EPC.

The mailbox registers may be accessed as byte, short, word, or multiple word quantities.

Figure 50: Block Diagram of Mailbox Registers



11.1.2 Doorbell Interrupts

Each of the 16 mailbox registers can generate four different interrupt requests called *doorbell interrupts*. Each of these requests can be independently masked for each mailbox register. The four doorbell interrupt types are:

- Local interrupt request on read from PCI side
- Local interrupt request on write from PCI side
- PCI interrupt request on read from Local side
- PCI interrupt request on write from Local side

The PCI read and Local read interrupts are OR'd together and latched in the mailbox read interrupt status register (MAIL_RD_STAT). Similarly, the PCI write and Local write interrupts are OR'd together and latched in the mailbox write interrupt status register (MAIL_WR_STAT). All of the interrupt request outputs from the status registers are OR'd together to form a single mailbox unit interrupt request and routed to both the Local and PCI Interrupt Control Units.

When a block of mailbox registers are accessed simultaneously, for example when 4 mailbox registers are read as a word quantity, then each register affected will request a separate interrupt if programmed to do so.

11.2 PROGRAMMING THE MAILBOX REGISTERS

After RESET, interrupt requests for all mailbox registers are disabled. The programmer must specifically enable interrupt requests for each mailbox register and for each access type.

11.2.1 Enabling Doorbell Interrupt Requests

The four types of doorbell interrupts described above are enabled in four 16-bit registers as shown in Table 13.

Table 13: Doorbell Interrupt Types and Corresponding Enable/Mask Registers

ACTION CAUSING INTERRUPT	REGISTER
PCI side read	PCI_MAIL_IERD
PCI side write	PCI_MAIL_IEWR
Local side read	LB_MAIL_IERD
Local side write	LB_MAIL_IEWR

11.2.2 Clearing Doorbell Interrupt Requests

All of the interrupt requests from the 16 mailbox registers are logically OR’d together and then forwarded to the Local Interrupt Control Unit (LICU) and the PCI Interrupt Control Unit (see Figure 50). For an interrupt handler to clear the local mailbox/doorbell interrupt request in the LICU, it must clear *all* of the enabled local mailbox interrupt requests from the individual mailbox registers. This is done by clearing the corresponding bit(s) in the MAIL_RD_STAT and MAIL_WR_STAT registers (by writing a ‘1’), where the mailbox interrupt requests are latched. Similarly, to clear the PCI mailbox/doorbell interrupt request in the PICU, it must clear *all* of the enabled PCI mailbox interrupt requests from the individual mailbox registers. This is done by clearing the corresponding bit(s) in the MAIL_RD_STAT and MAIL_WR_STAT registers, where the mailbox interrupt requests are latched.

Note that MAIL_RD_STAT and MAIL_WR_STAT registers are cleared by writing a ‘1’ into the bits to be cleared. Writing a ‘0’ will have no effect. Interrupt status bits in the Local Interrupt Control Unit (LB_ISTAT) and the PCI Interrupt Control Unit (PCI_INT_STAT) are cleared by writing a ‘0’ and are unaffected by writing a ‘1’.

Chapter 12 Interrupt Control

The EPC includes two Interrupt Control Units (ICUs): one to process interrupt requests bound for the local CPU, and one to process interrupt requests to and from the PCI bus. The two Interrupt Control Units are connected to allow the routing of interrupt requests from PCI to the Local bus, as well as from the Local bus to PCI.

The PCI Interrupt Control Unit (PICU) also includes a special *crosspoint interrupt routing mechanism* for the four PCI interrupt requests ($\overline{\text{INTA}}$ through $\overline{\text{INTD}}$). Each of the four PCI interrupts can function as either an input or an output, and interrupt requests may be passed from any PCI interrupt to any other PCI interrupt. PCI interrupt acknowledge cycles can also be generated by the EPC when acting as a host bridge.

12.1 LOCAL INTERRUPT CONTROL UNIT

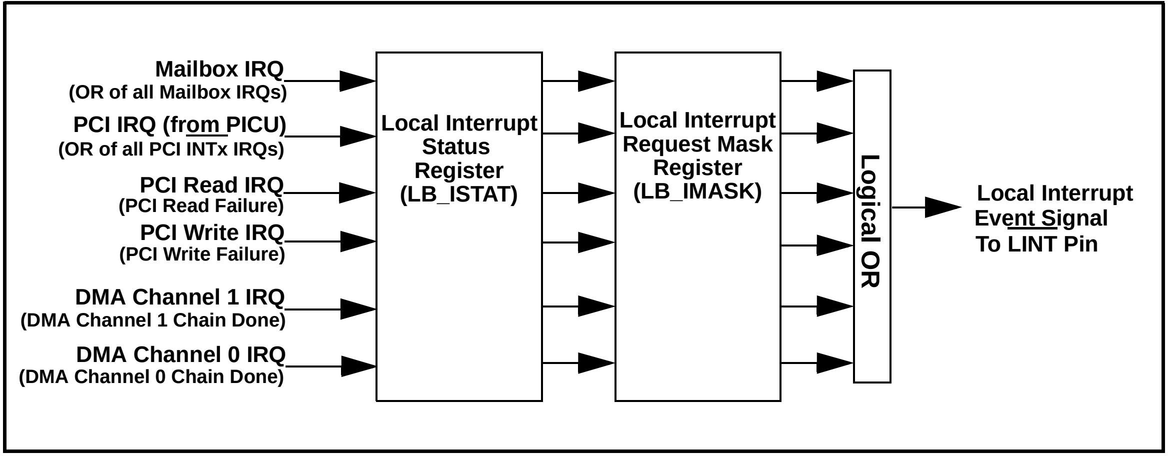
The Local Interrupt Control Unit (LICU) process interrupts from the following sources:

- PCI read and write exceptions
- DMA chain completion
- Mailbox register access (“doorbell” interrupt)
- PCI Interrupt pin events

12.1.1 Overview

The LICU consists of an interrupt status register (LB_ISTAT) and an interrupt mask register (LB_IMASK) as shown in Figure 51. Interrupt requests from the individual sources are posted in the interrupt status register. The interrupt mask register controls which of the interrupt requests posted in the request register actually generate an interrupt request to the local processor, and optionally, the PCI Interrupt Control Unit.

Figure 51: **Local Interrupt Control Unit Block Diagram**


Mailbox IRQ
(OR of all Mailbox IRQs)
PCI IRQ (from PICU)
(OR of all PCI INTx IRQs)
PCI Read IRQ
(PCI Read Failure)
PCI Write IRQ
(PCI Write Failure)
DMA Channel 1 IRQ
(DMA Channel 1 Chain Done)
DMA Channel 0 IRQ
(DMA Channel 0 Chain Done)
Local Interrupt Status Register (LB_ISTAT)
Local Interrupt Request Mask Register (LB_IMASK)
Logical OR
Local Interrupt Event Signal To LINT Pin

12.1.2 Local Interrupt Requests

The following local interrupt requests are physically latched within the LB_ISTAT register:

- "PCI_RD", "PCI_WR" bits: PCI Read and Write Error Interrupts
- "DMA0", "DMA1" bits: DMA Channels 0 and 1 Interrupts
- "MAILBOX" bit: Mailbox interrupt requests[1]

The above interrupt requests are cleared by clearing (0) the corresponding bits in the LB_ISTAT register.

The following local interrupts are not latched within the LB_ISTAT register:

- "PCI_INT" bit: PCI interrupt control unit requests

Non-latched interrupt requests are only cleared when the source of the corresponding interrupt request is cleared. For example, the PCI_INT request is only cleared when *all* enabled PCI interrupt inputs are de-asserted.

12.1.3 Masking Local Interrupt Requests

Local interrupt requests can be masked (disabled) by clearing the corresponding bit in the LB_IMASK register. The interrupt mask register controls (enables) which of the interrupt requests posted in the request register actually generate an interrupt request.

12.1.4 Local Interrupt Event Signal

The LICU has a single output signal that is the logical OR of all unmasked interrupt requests.

[1] The MAILBOX bit is not latched in silicon revision B2 or later.

This signal is routed to the local processor interrupt pin ($\overline{\text{LINT}}$) and to the PCI Interrupt Control Unit. The activation of the LICU output signal in response to an interrupt is called a *local interrupt event*.

12.2 PCI INTERRUPT CONTROL UNIT (PICU)

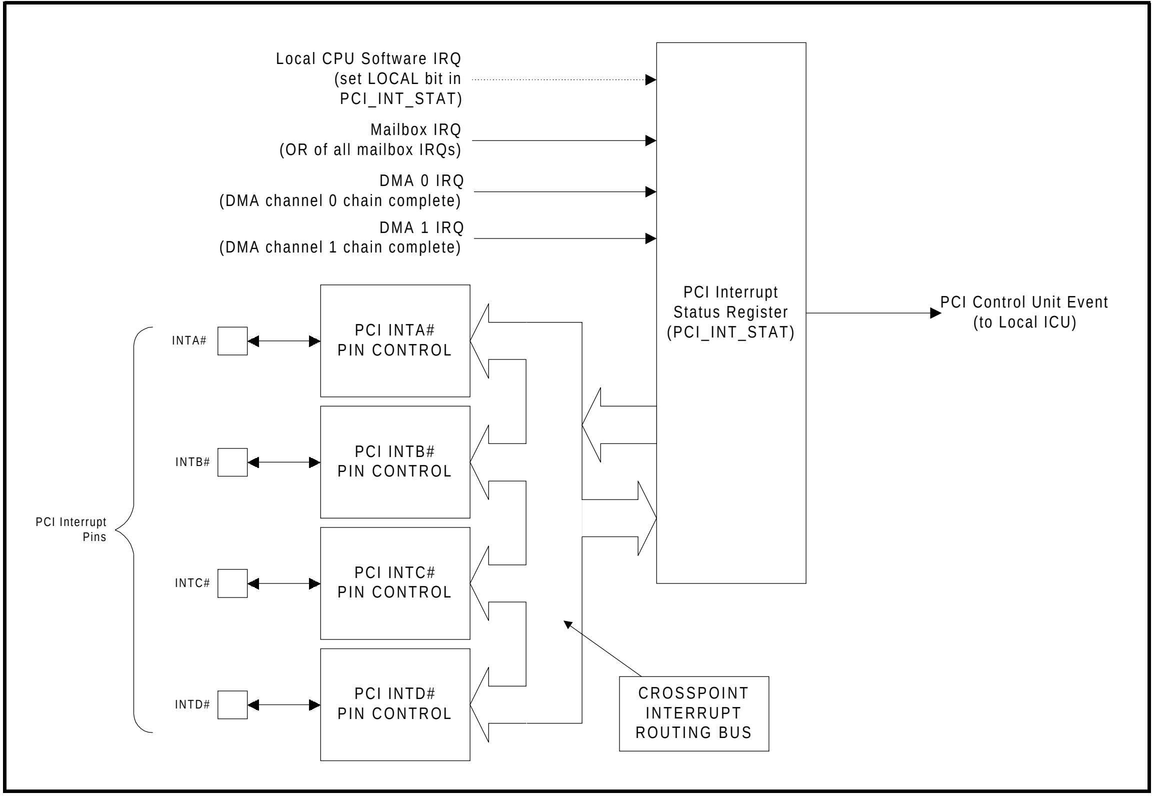
The PCI Interrupt Control Unit (PICU) process interrupts from the following sources:

- DMA Chain Completion
- Mailbox Register Access (“Doorbell” interrupt)
- Local bus direct software interrupts
- Interrupt requests from the $\overline{\text{INTA}}$, $\overline{\text{INTB}}$, $\overline{\text{INTC}}$, and $\overline{\text{INTD}}$ PCI interrupt request pins (when configured as inputs)

12.2.1 Overview

The PICU is significantly more complex than the LICU, as shown in Figure 53. The interrupt crosspoint mechanism allows for maximum system design flexibility for embedded systems using multiple PCI interrupts. In addition, control is provided for both receiving and requesting interrupts on any of the four PCI interrupt pins.

Figure 52: PCI Interrupt Control Unit Block Diagram



12.2.2 PCI Interrupt Pins (INTA through INTD)

The PCI Specification provides for four shared PCI interrupt request pins: INTA through INTD. Typically, these interrupts are driven by the target devices in a system and received by the host bridge device. Since the EPC can be used as either a host or target bridge, the capability is provided to either receive interrupts or drive interrupt requests on the INTA through INTD pins.

12.2.2.1 Configuring a PCI Interrupt Pin as an Interrupt Request Output

Any of the INTx pins can be configured as an output pin. However, only *one* of the INTx pins can be designated as the destination for mailbox, local direct, and DMA interrupt requests (chosen by the INT_PIN field in the PCI_BPARAM register). The remaining INTx outputs can be programmed to reflect the state of other INTx pins configured as *inputs* (see Crosspoint Routing Mechanism, below).

The direction of the INTx pins is set via the MODEx fields in the PCI Interrupt Configuration register (PCI_INT_CFG). Pins configured as outputs are always active low and are software cleared. Software cleared outputs will go inactive when the corresponding bit in the PCI_INT_STAT register is cleared.

12.2.2.2 Configuring a PCI Interrupt Pin as an Interrupt Request Input

Any of the INTx pins may be configured as inputs. The direction and detection mechanism for INTx pins is controlled by the MODEx field in the PCI_INT_CFG register. INTx inputs may be either edge or level sensitive. Edge sensitive pins will generate an interrupt event when a high-to-low transition is seen on the pin. Level triggered inputs will generate an interrupt event whenever the state of the INTx pin is low. (Note that level triggered inputs will generate another interrupt event if a previous request is cleared AND the corresponding INTx pin is still at a logic “0”.)

PCI interrupt inputs will post interrupt requests in the PCI_INT_STAT register *only when they are routed to a PCI interrupt output* through the crosspoint routing mechanism (see below).

12.2.2.3 Crosspoint Interrupt Routing Mechanism

Many embedded designs will require the ability to control multiple PCI interrupt input and output events. The EPC’s crosspoint interrupt routing mechanism allows for extremely flexible interrupt request routing between the four interrupt pins.

For example, consider the case of an expandable network router using PCI as the backplane. In such a system there may be no single “host processor”. Each plug in board may act as a host processor from time to time, and will need the ability to both receive interrupt requests as well as post them. It would be nice if such a system could be designed to be “plug and play”, so that when a new board was added, it automatically configured some of its interrupts as inputs, and some as outputs *dynamically*. The interrupt crosspoint routing mechanism allows the system designer to do just that.

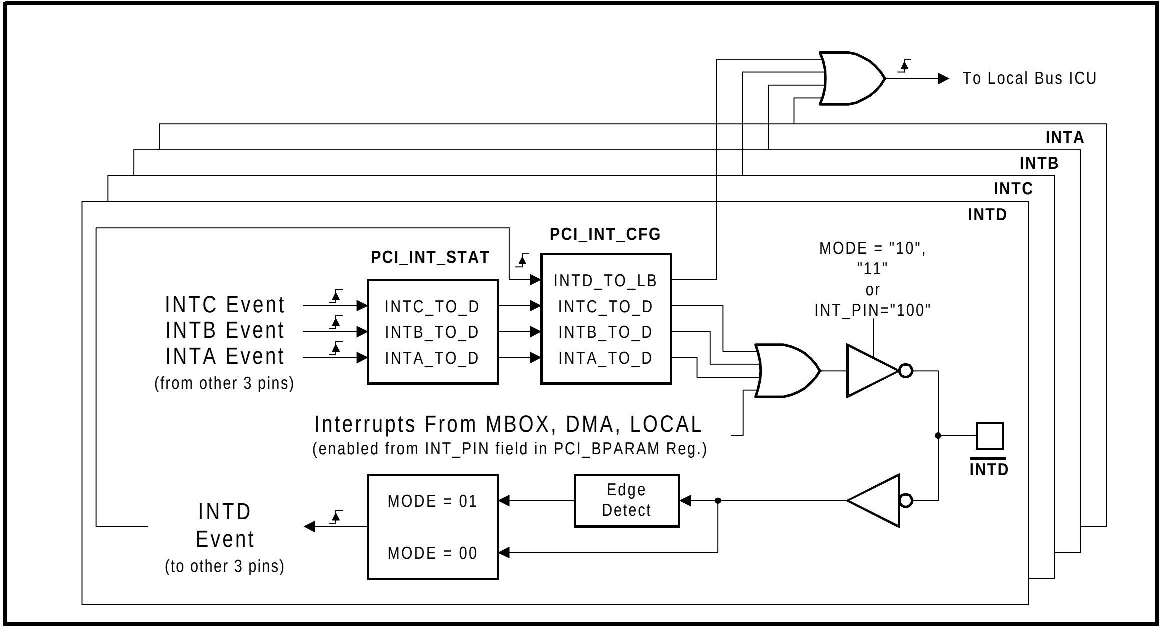
Figure 53 shows the details of the crosspoint routing mechanism. Each INTx pin is capable of generating a PCI interrupt event, which is in turn routed to all of the other INTx pin control circuits. When an INTx pin is configured as an output, it can be driven internally from either one or more of the other 3 INTx pin interrupt events, or from the combination of the mailbox, DMA, and local direct source.

The routing of interrupts is controlled through the INTx_TO_y bits in the PCI_INT_CFG register. For example, if the INTA_TO_D bit is set *and* INTA is configured as an input pin *and* INTD is configured as an output, then when INTA is active INTD will go low to request an interrupt on the PCI bus.

The individual INTx interrupt requests may also be routed to the Local Interrupt Control Unit. Each pin has an associated INTx_TO_LB bit that selects it as an input to the LICU. All PCI interrupt pin requests are OR'd before presentation to the LICU (see Figure 53).

PCI interrupt configuration is shown in more detail in the programming example below.

Figure 53: Interrupt Crosspoint Routing Mechanism



12.2.3 Internal PCI Interrupt Requests

In addition to the INTx pins, the PICU can also respond to request from the mailbox registers, the DMA controller, and the local CPU. This group of PCI interrupt request sources is called *internal PCI interrupt requests* to differentiate them from the INTx pins.

12.2.3.1 Mailbox and DMA PCI Interrupt Requests

Like the local Interrupt Control Unit, the PCI Interrupt Control Unit can receive interrupt requests from both the mailbox registers and the DMA controller. Also like the LICU, the PICU latches the DMA interrupt requests and *does not latch* the mailbox interrupt request.

Clearing the DMA interrupt requests is achieved by writing "1" to the corresponding bit in the PCI_INT_STAT register. Note that clearing a DMA interrupt request in PCI_INT_STAT has no affect on the DMA interrupt request bits in the LB_STAT register.

Mailbox interrupt requests are only cleared by clearing the individual mailbox interrupt requests in the mailbox register unit.

12.2.3.2 Local Direct Interrupt Request

The Local processor may request a PCI interrupt by setting the LOCAL bit in the PCI_INT_STAT register. The LOCAL bit can be cleared from the PCI or Local side of the bridge.

12.2.3.3 Routing the Internal PCI Interrupt Requests to an $\overline{\text{INTx}}$ Pin

Only one $\overline{\text{INTx}}$ pin can be the destination for internal PCI interrupts. The specific pin is determined by programming the INT_PIN field in the PCI_BPARAM register. The pin used for this purpose may also be used as the destination for crosspoint routed $\overline{\text{INTx}}$ interrupts.

12.2.4 PICU Configuration Example

An example may be helpful in describing the operation of the PCI Interrupt Control Unit. Figure 54 shows a system block diagram for the following example. The interrupt pin usage is detailed in Table 14.

First, we'll need to mask DMA and mailbox interrupts from appearing on the $\overline{\text{INTC}}$ pin when we “turn it on” as an output (this will prevent spurious interrupts from occurring). These interrupts are disabled by clearing the MAILBOX and DMA0/1 bits in the PCI_INT_CFG register.

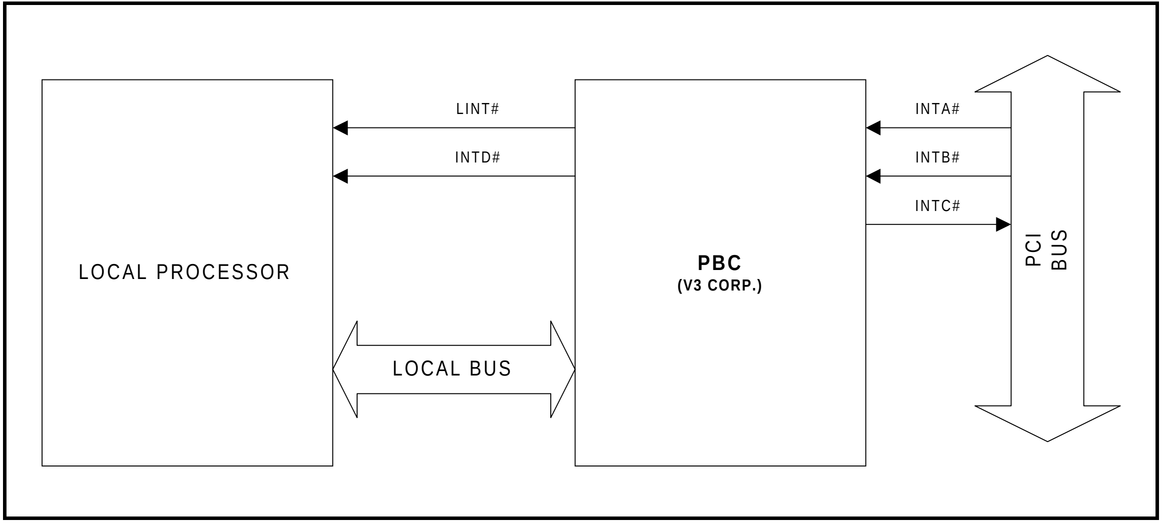
The next step is to configure $\overline{\text{INTD}}$ and $\overline{\text{INTC}}$ as outputs. This is done by setting the appropriate mode in the MODED and MODEC fields in the PCI_INT_CFG register. We'll choose “software cleared” for both (MODEx=10). Since $\overline{\text{INTC}}$ needs to be able to drive the local direct interrupt onto the PCI bus, we must also set INT_PIN field in the PCI_BPARAM register to “011” (this sets $\overline{\text{INTC}}$ as the “receiver” for internal PCI interrupts). Because all of the PCI interrupts are open drain (by PCI definition), we'll need to make sure the hardware guys put a pullup resister on the $\overline{\text{INTD}}$ pin that's used as a local interrupt (it's not on the block diagram).

Finally, we need to configure the $\overline{\text{INTA}}$ and $\overline{\text{INTB}}$ pins as inputs to $\overline{\text{INTD}}$ (the PCI interrupt routing request pin). This is done by setting the INTA_TO_D and INTB_TO_D bits in the PCI_INT_CFG register. Configuration is now complete.

Table 14: Example PCI Interrupt Usage

Interrupt Pin	Use
$\overline{\text{LINT}}$	Tied to one of the local CPU's interrupt input pins. Signals Local interrupt requests, including PCI errors (very high priority).
$\overline{\text{INTD}}$	Also tied to one of the local CPU's interrupt input pins. This pin is configured as an output and forwards PCI interrupt requests from the $\overline{\text{INTA}}$ and $\overline{\text{INTB}}$ pins to the local CPU (medium priority).
$\overline{\text{INTC}}$	Configured as an output, $\overline{\text{INTC}}$ is used to signal local direct processor requests to other subsystems in the PCI bus.
$\overline{\text{INTA}}$ and $\overline{\text{INTB}}$	Generic PCI interrupt inputs from target subsystems.

Figure 54: System Block Diagram for PCI Interrupt Control Unit



12.3 GENERATING PCI INTERRUPT ACKNOWLEDGE CYCLES

Generating PCI interrupt acknowledge cycles is straightforward with the EPC:

- Set the TYPE field in one of the LB_MAPx registers to "Interrupt Acknowledge".
- Turn off prefetching for the corresponding aperture by clearing the PREFETCH bit in the LB_BASEx register.
- Perform a single word read access to the aperture. The data returned will be the interrupt vector.

Chapter 13 Initialization

Initialization of the EPC usually occurs when the system is reset. The EPC can be reset either from the PCI bus or from the local bus. Following a reset, the EPC's internal registers can be initialized via the PCI bus, the local bus, or the serial EEPROM interface. Internal registers can be modified after reset from the PCI bus (memory, I/O, or configuration space) or from the local bus. This chapter describes reset options and register initialization procedures.

13.1 RESET DIRECTION

The EPC device is reset by driving active either the $\overline{\text{LRST}}$ pin (for resetting from the local bus) or $\overline{\text{PRST}}$ pin (for resetting from the PCI bus). Which pin is used as the EPC reset input is controlled by the RDIR pin.

The EPC is also capable of generating reset for either the local or the PCI bus. For systems using the EPC as the PCI system master bridge, PCI reset is typically driven by the EPC. Systems using the EPC as a target bridge will typically receive reset via the PCI bus, and reset the host processor in turn. Figure 55 shows examples of both reset directions.

Eight clock cycles are required for both the PCI and local bus following the rising edge of reset before attempting to access the internal registers. This eight clock delay is necessary to allow the internal operations of the EPC to achieve an idle state.

Figure 55: Reset Direction Examples

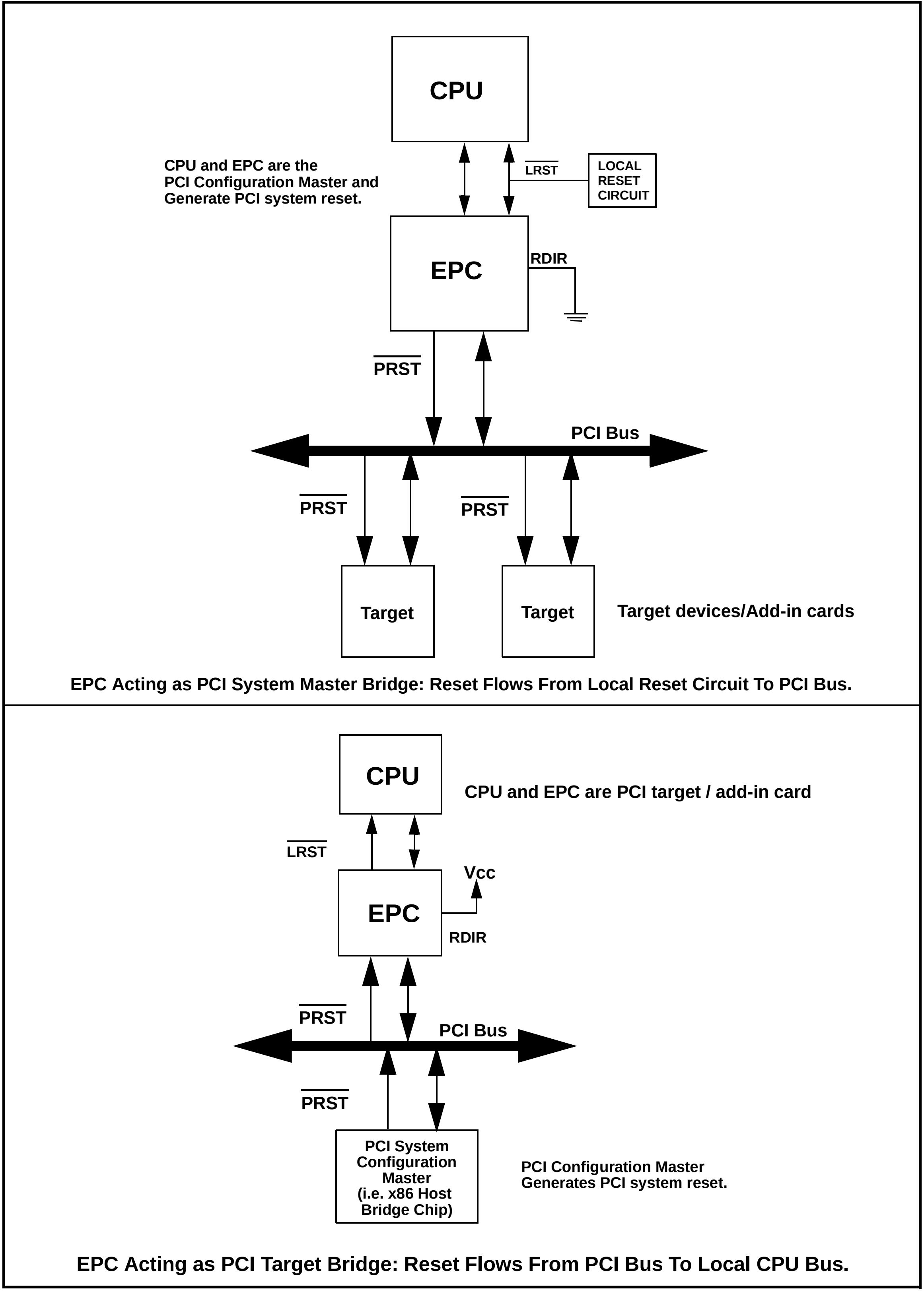

CPU
CPU and EPC are the
PCI Configuration Master and
Generate PCI system reset.
LRST
LOCAL
RESET
CIRCUIT
EPC
RDIR
PRST
PCI Bus
PRST
PRST
Target
Target
Target devices/Add-in cards
EPC Acting as PCI System Master Bridge: Reset Flows From Local Reset Circuit To PCI Bus.
CPU
CPU and EPC are PCI target / add-in card
LRST
Vcc
EPC
RDIR
PRST
PCI Bus
PRST
PCI System
Configuration
Master
(i.e. x86 Host
Bridge Chip)
PCI Configuration Master
Generates PCI system reset.
EPC Acting as PCI Target Bridge: Reset Flows From PCI Bus To Local CPU Bus.

Table 15: RESET Direction Options

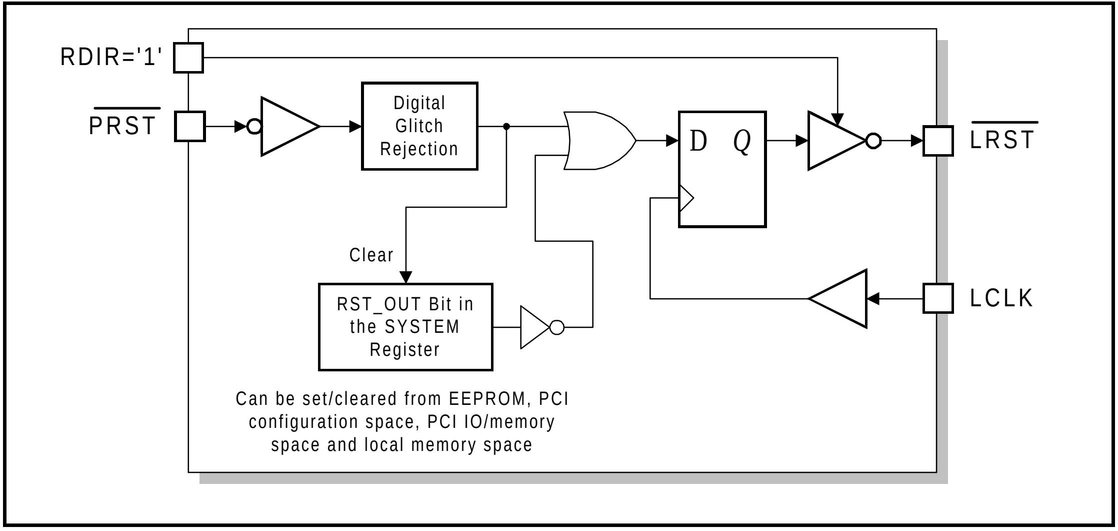
RDIR Pin State	Direction	$\overline{\text{LRESET}}$	$\overline{\text{PRST}}$	Comments
0	Local to PCI	Input	Output	$\overline{\text{PRST}}$ is deasserted by writing the RST_OUT bit in the SYSTEM register
1	PCI to Local	Output	Input	$\overline{\text{LRST}}$ is deasserted by writing the RST_OUT bit in the SYSTEM register

The reset used as output remains asserted until the input Reset has been de-asserted and the RST_OUT bit in the SYSTEM register has been written “1”. The RST_OUT bit can be written ‘0’ later to cause the output Reset to be asserted again (software controlled Reset).

The following diagram illustrates how the EPC reset operation works with the RDIR pin tied high. In this mode the PCI bus provides the reset input via $\overline{\text{PRST}}$. This $\overline{\text{PRST}}$ input is processed to produce a local Reset output $\overline{\text{LRST}}$.

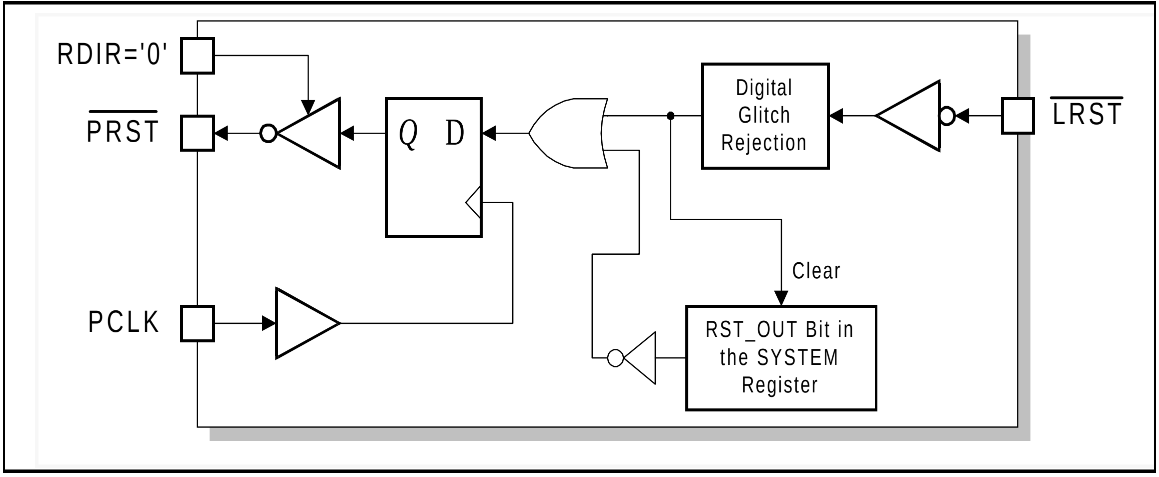
In many systems, it will be desirable to un-reset the local bus after the PCI bus has been released from reset. This is accomplished automatically by programming the serial EEPROM with bit 7 set in byte 79H (this is the RST_OUT bit in the SYSTEM register).

Figure 56: Reset Operation when RDIR = 1


RDIR='1'
$\overline{\text{PRST}}$
Digital Glitch Rejection
D
Q
$\overline{\text{LRST}}$
Clear
RST_OUT Bit in the SYSTEM Register
LCLK
Can be set/cleared from EEPROM, PCI configuration space, PCI IO/memory space and local memory space

By tying the RDIR pin low, the Reset direction can be configured to drive out the PCI reset and use $\overline{\text{LRST}}$ as an input.

Figure 57: Reset Operation when RDIR = 0



13.2 INITIALIZING THE INTERNAL REGISTERS

There are three options for initializing the EPC internal registers set after reset:

- Initialization from the local bus side (through the aperture defined by the LB_IO_BASE register)
- Initialization from the PCI bus side via configuration space
- Initialization from the serial EEPROM interface

13.2.1 Selecting Initialization Mode

There are 3 initalization modes that can be selected to control the operational of the EEPROM controller during initalization (see Table 16).

Table 16: EEPROM Initalization Options

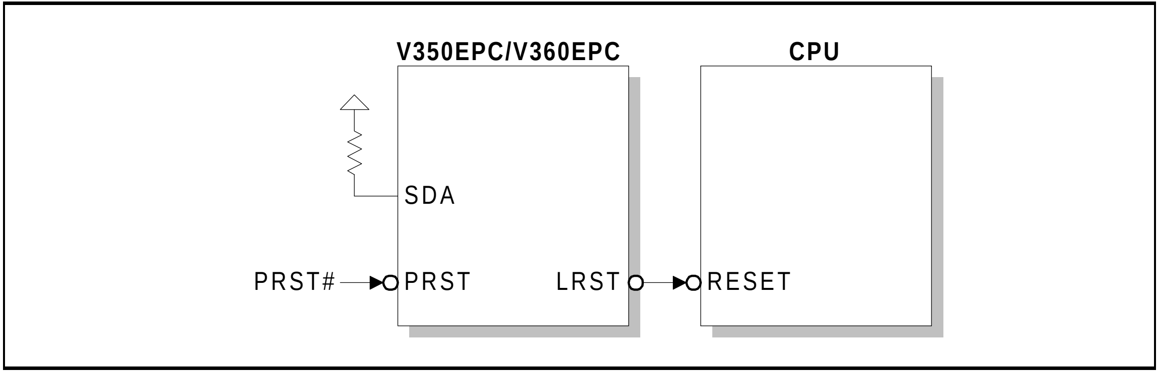
EEPROM Port Connection	RETRY_EN	RST_OUT	Comment
SDA pulled high	1	1	Typically used for initalization via local processor
SDA Tied Low	0	0	Typically used for initalization via PCI
SDA and SCL connected to valid EEPROM device	From EEPROM	From EEPROM	Initalization from EEPROM

After the input reset has been released, 10 SCL clocks are performed followed by a STOP cycle to ensure that a re-reset didn’t falsley see the SDA signal low.

13.2.2 Initialization Using the Local Processor

In applications where the EPC is a secondary master it may be desirable to use the local processor to initialize the contents of the internal registers of the EPC and save the cost of the serial EEPROM device. This is best accomplished by pulling the SDA pin high on the EPC and using PRST# as an input. The processor can be connected as in Figure 58.

Figure 58: Connection for Initialization Using the Local Processor



When connected as in Figure 58, the RETRY_EN bit of the PCI_CFG register will be set when PRST# is asserted and remain that way until the local CPU clears it. RETRY_EN is used to cause a PCI configuration access to the EPC to get retried until it is cleared from the local CPU. This ensures that the local bus CPU gets a chance to properly initialize before the PCI BIOS tries to enumerate the EPC. Once the local CPU has initalized the internal registers then RETRY_EN can be cleared to allow the primary PCI master to enumerate the EPC.

Access to the internal registers from the local bus side is through the “local bus-to-internal register” aperture defined by the LB_IO_BASE register. LB_IO_BASE is a sixteen bit register which defines a field that is compared with address bits A[31:16] for *every access* seen on the local bus. When there is a match between LB_IO_BASE and A[31:16], that access is “claimed” by the EPC and is considered to be to an access to the internal registers. The local-to-internal register aperture has a granularity of 64Kbytes (since only A[31:16] are compared) even though only the first 256 bytes are used. The remaining space is reserved.

As an example, let’s assume LB_IO_BASE is programmed to the value 1234H. To read the PCI_CC_REV register (offset 08H) you would perform a word read from location 1234.0008H in local memory. Care must be taken when using big-endian processors (or big-endian regions with little-endian processors) to generate the proper addresses, as configuration space is little-endian (by PCI definition).

In order for the LB_IO_BASE decoder to respond to a local bus cycle at the desired memory location, it must be configured. There are 3 ways to do this:

- LB_IO_BASE can be downloaded from the serial EEPROM device
- LB_IO_BASE can be configured from the PCI bus through configuration space or through the PCI_IO_BASE register

- LB_IO_BASE can be configured by the local processor by capturing the first write cycle on the local bus

After RESET, the local bus interface on the EPC will respond to any write cycle it sees on the bus until the LB_IO_BASE register is written (regardless of how it is written). Therefore, if the serial EEPROM download is enabled, then the contents of LB_IO_BASE is established in this way and the EPC will only respond to addresses in the range determined by LB_IO_BASE. The same is true of initialization from the PCI bus.

If initialization of LB_IO_BASE is to be performed by the local processor then it should be done as the first (or one of the first) write cycle. The address should be xxxx.006Ch[1] where xxxx is the base address that internal EPC registers are to be mapped in local space. The data for the write should also be the base address (this is the data that will actually be written into the register). If the first write is NOT to location xxxx.006Ch then the EPC will also claim the access and write one of the internal registers (the low address determines which register is selected). When xxxx.006Ch is eventually written then the EPC will only respond at the location determined by the value loaded into LB_IO_BASE. For the V960EPC with it’s 16 bit bus, LB_IO_BASE is initialized with a single write to xxxx.006Eh which is the upper (and only significant) part of the LB_IO_BASE register. In summary, when the EPC is initialized by the local bus processor (as opposed to PCI bus or serial EEPROM):

- The EPC will capture all write cycles on the local bus until the LB_IO_BASE register (at offset 6Ch) is written by something (the Serial EEPROM, PCI bus or local bus processor)

- Writes to addresses other than xxxx.006Ch will also be captured by the EPC.

- All cycles that the EPC captures will generate a “data ready” to acknowledge the transfer

Once the LB_IO_BASE register is set, the remaining internal registers are programmed by writing to the corresponding memory mapped locations. Burst reads and writes are permitted to internal registers.

All PCI and local bus functions are disabled following a hardware reset. For example, all PCI accesses are ignored by the EPC until enabled by the programmer (with the exception of PCI configuration cycles as described below).

.

Table 17: Local Bus Signals for LB_IO_BASE Configuration after RESET

CPU	READ/ WRITE SIGNAL	$\overline{\text{BE3}}$ / BWE3	$\overline{\text{BE2}}$ / BWE2	$\overline{\text{BE1}}$ / BWE1	$\overline{\text{BE0}}$ / BWE0	ADDRESS	DATA
i960Jx i960Cx i960Hx	WRITE	0	0	0	0	xxxx.006CH	D[31:16] = LB_IO_BASE D[15:0] = 006CH

[1] The LB_IO_BASE register is in the upper 16 bits for a 32-bit write. Address 6Eh should be used for a 16-bit write using a little endian processor.

Table 17: Local Bus Signals for LB_IO_BASE Configuration after RESET (cont’d)

CPU	READ/ WRITE SIGNAL	$\overline{\text{BE3}}$ / BWE3	$\overline{\text{BE2}}$ / BWE2	$\overline{\text{BE1}}$ / BWE1	$\overline{\text{BE0}}$ / BWE0	ADDRESS	DATA
i960SA	WRITE	NA	NA	0	0	xxxx.006EH xxxx.006CH	D[15:0] = LB_IO_BASE D[15:0] = 006CH (2 cycles)
Am2903x Am2904x	WRITE	0	0	0	0	xxxx.006CH	D[31:16] = LB_IO_BASE D[15:0] = 006CH

13.2.2.1 i960 Processor Configuration Note

In order for the V96xEPC to interact correctly with the i960 family of processors, the following processors parameters should be used:

- Burst - Enable
- External Ready - Enable
- Pipelining - Disable
- N_{RAD} - 0
- N_{RDD} - 0
- N_{WAD} - 0
- N_{WDD} - 0
- N_{XDA} - 0
- Bus Width - 32 Bits

13.2.3 Initialization Using the PCI Configuration Space

The EPC can be initialized from the PCI bus by a PCI system master via configuration space accesses. Typically, this type of configuration would be used in systems using the EPC as a target bridge for a PCI add-in card.

Access to the configuration space of the EPC from the PCI bus is achieved by asserting the EPC’s IDSEL pin. With IDSEL active, PCI transfers in the address range 0-FFH will be forwarded to the internal configuration registers. The EPC ignores Type 1 configuration accesses (PCI to PCI bridge configuration cycles).

13.2.4 Initialization Using the Serial EEPROM interface

The EPC registers can also be initialized using the serial EEPROM interface. When using this method, the EPC “downloads” the values for the first half of the 256-byte internal register block from a serial EEPROM connected to the SCL and SDA pins. All register bits of type “FRW” or “FR” for configuration registers from 00H to 7FH are written by the serial EEPROM. Only bits of type “R” cannot be written.

Serial EEPROM initialization is useful in the following situations:

- Systems without a local processor that are using the EPC as a stand alone bridge
- Systems that cannot guarantee that the EPC can be initialized in a timely fashion by the local processor
- Systems which wish to eliminate an additional variable (e.g. during software debug)

The EPC uses a I^2C-like interface to download from the serial EEPROM. The interface is designed to work with 24C02 style serial EEPROMs. Table 18 shows serial EEPROMs known to be compatible with this interface.

When the reset input ($\overline{\text{LRESET}}$ or $\overline{\text{PRST}}$, as selected by RDIR) is de-asserted, the SDA pin is sampled at the rising edge of the reset input. If a serial PROM is present, the SDA pin will be pulled high by the external pull up resistor. The detection of a high signal on SDA at the rising edge of the reset input begins the serial download. Each byte is read from the EEPROM sequentially, starting from 0H and ending at 7FH.

NOTE*: To prevent serial initialization for systems initializing from the PCI or local bus*, YOU MUST TIE SDA TO GROUND through a 1-2.2K ohm resistor. See Figure 59 for example circuits.

Table 18: Serial EEPROMs Known Compatible with the EPC

Manufacturer	Device
Atmel	24C02
Microchip	24C02A
Xicor	X24C02

13.2.4.1 Programming the Serial EEPROM

Programming of the serial EEPROM is straightforward on most commercial EPROM programmers. Each byte in the 128-byte EEPROM is programmed with the value desired in the corresponding byte in the EPC's register map. Unused registers should be programmed with 0H.

The EPC may be used to program the EEPROM in the target application. Direct control over the state of the SCL and SDA pins is available through the SYSTEM register. The system programmer must provide the proper signal timing for the serial EEPROM through software emulation of the I^2C-like protocol. Examples of source code for programming the I^2C protocol can be found on the World Wide Web by searching for I^2C using one of the internet search engines.

ApNote: If the VENDOR ID and DEVICE ID fields are programmed to FFFFH (an illegal value) in the EEPROM, the EPC will ignore these values and leave the default power on values in these registers.

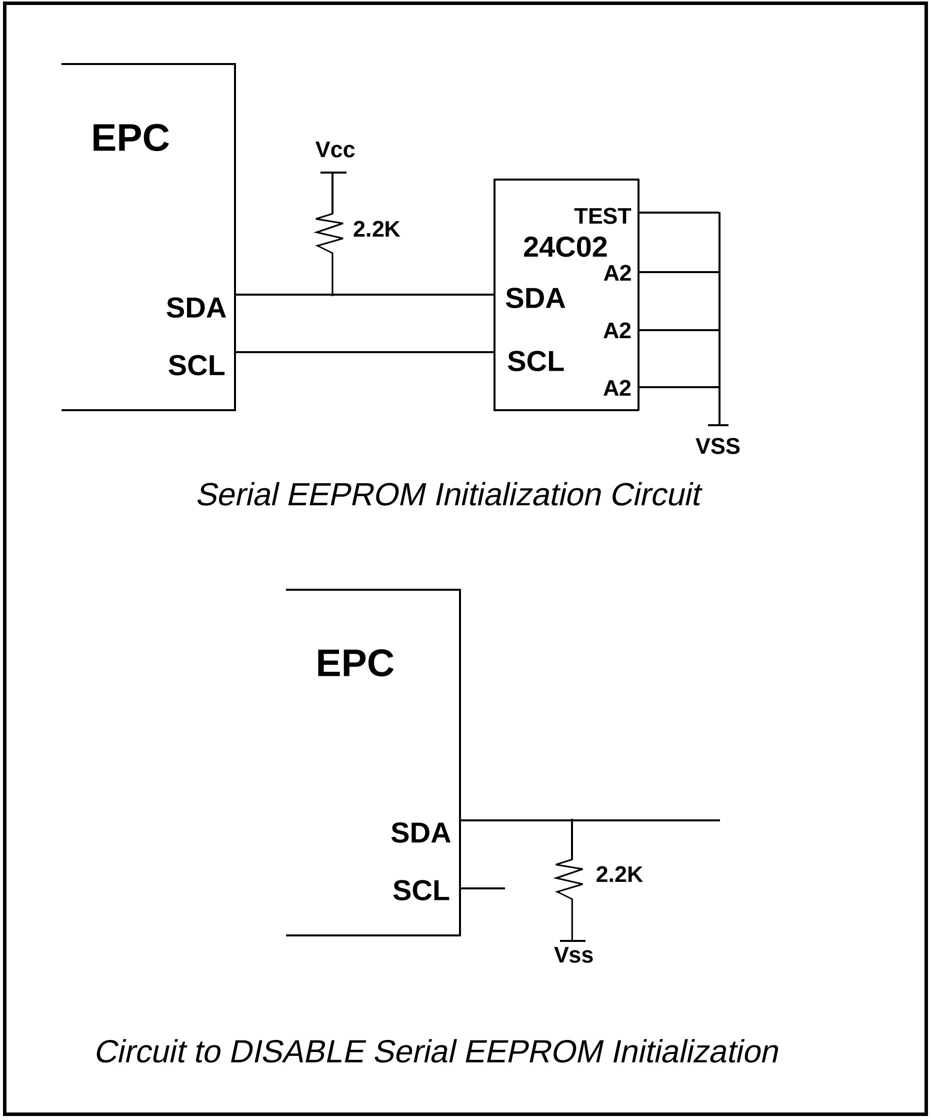
13.2.4.2 Timing Considerations when Initializing via the Serial EEPROM

The download of the serial EEPROM data takes a considerable amount of time. Each byte of data requires 9 SCL cycles at 512 PCI clocks per cycle. There is additional set-up overhead of approximately 5 data bytes (for serial EEPROM setup). The timing for a full download is:

(512 PCI clocks) x (133 bytes) x (9 SCL's per byte) = 613,000 PCI clocks

Access from PCI will cause Retry until the download is done.

Figure 59: Serial EEPROM Initialization Schematics



Serial EEPROM Initialization Circuit

Circuit to DISABLE Serial EEPROM Initialization

13.2.5 Re-Initialization Using the PCI I/O or Memory Space

The EPC’s internal registers may also be accessed via the PCI-to-Internal register aperture. The base address for this aperture is controlled by the PCI_IO_BASE register. This method can not be used to initialize the EPC immediately following a RESET (unless a serial EEPROM is used) because PCI memory reads and writes are ignored following RESET until one or both of the MEM_EN and IO_EN bits in the PCI_CMD register are set (’1’) and the PCI_IO_BASE register base address has been established.

Chapter 14 Register Descriptions

The registers for the EPC are broken into 6 basic groups: System, PCI Configuration and Control, Local Configuration and Control, FIFO Configuration, DMA Control, and Mailbox Registers. This chapter will describe the location and attributes of the registers; programming details for each register are given in the appropriate chapters.

Each bit in the EPC registers is readable and writable according to one of the following designations. Those marked with an asterisk (*) apply to the PCI Configuration Registers only and comply with the PCI specification to provide the required PCI configuration header.

R: Read only - bits are internally driven and cannot be modified.

FR*: Firmware Initialized, Configuration Read Only - these bits are initialized after a system reset by downloading via the serial EEPROM device or by the local bus master. Once "FR" bits are loaded they may be locked from further modification by setting the LOCK bit in the SYSTEM register.

W: Write only. Typically used to issue commands.

FRW*: Firmware Initialized, Configuration Read/Write - Initialized at boot-time but can be both read or written from the PCI and Local buses.

RW: Read and Write.

All reserved register bits read back as zeros.

The PCI configuration registers required by the PCI Spec are described in the PCI Configuration Registers section below.

14.1 REGISTER MAP

Figure 60 shows the internal register map for the EPC. The offsets shown are relative to the base address of the aperture from which the registers are accessed:

- LB_IO_BASE for accesses from the local bus
- PCI_IO_BASE for memory or I/O accesses from the PCI bus
- 0H in configuration space for configuration accesses from the PCI bus

Figure 60: **Register Map**

REGISTER				OFFSET
31	16	15	0	
PCI_DEVICE		PCI_VENDOR		00H
PCI_STAT		PCI_CMD		04H
PCI_CC_REV				08H
PCI_HDR_CFG				0CH
PCI_IO_BASE (PCI_I20_BASE when I20 operation is enabled: I20_EN bit)				10H
PCI_BASE0				14H
PCI_BASE1				18H
reserved				1C-2BH
PCI_SUB_ID		PCI_SUB_VENDOR		2CH
PCI-ROM				30H
reserved				34-38H
PCI_BPARAM				3CH
PCI_MAP0				40H
PCI_MAP1 (PCI_I20_MAP[a] when I20 operation is enabled: I20_EN bit)				44H
PCI_INT_STAT				48H
PCI_INT_CFG				4CH
reserved				50H
LB_BASE0				54H
LB_BASE1				58H
LB_MAP0		reserved		5CH
LB_MAP1		reserved		60H
LB_MAP2		LB_BASE2		64H
LB_SIZE[b]				68H
LB_IO_BASE		reserved		6CH
FIFO_PRIORITY		FIFO_CFG		70H
LB_IMASK	LB_ISTAT	FIFO_STAT		74H
LB_CFG		SYSTEM		78H
reserved		PCI_CFG		7CH
DMA_PCI_ADDR0				80H
DMA_LOCAL_ADDR0				84H
DMA_CSR0	DMA_LENGTH0			88H
DMA_CTLB_ADR0				8CH
DMA_PCI_ADDR1				90H
DMA_LOCAL_ADDR1				94H
DMA_CSR1	DMA_LENGTH1			98H
DMA_CTLB_ADR1				9CH
reserved				A0H - BCH
MAIL_DATA3	MAIL_DATA2	MAIL_DATA1	MAIL_DATA0	C0H
MAIL_DATA7	MAIL_DATA6	MAIL_DATA5	MAIL_DATA4	C4H
MAIL_DATA11	MAIL_DATA10	MAIL_DATA9	MAIL_DATA8	C8H
MAIL_DATA15	MAIL_DATA14	MAIL_DATA13	MAIL_DATA12	CCH
PCI_MAIL_IERD		PCI_MAIL_IEWR		D0H
LB_MAIL_IERD		LB_MAIL_IEWR		D4H
MAIL_RD_STAT		MAIL_WR_STAT		D8H
QBA_MAP				DCH
			DMA_DELAY	E0H

PCI_VENDOR: VENDOR ID (PCI REQUIRED)

Mnemonic:	PCI_VENDOR
Offset:	00H
Size:	16 bits

PCI_VENDOR

Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15-0	VENDOR	FR	11B0H	Vendor ID. This register identifies the vendor of the device.

PCI_DEVICE: DEVICE ID (PCI REQUIRED)

Mnemonic:	PCI_DEVICE
Offset:	02H
Size:	16 bits

PCI_DEVICE

Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15-0	DEVICE	FR	see text	Device ID. This register identifies the ID of the device. At reset it reads back a value dependent on the type of EPC device: 960mode=01h, 961mode=02h, 962mode=04h, 292mode=10h

PCI_CMD: COMMAND REGISTER (PCI REQUIRED)

Mnemonic:	PCI_CMD
Offset:	04H
Size:	16 bits

PCI_CMD

Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15-10	-	R	0H	reserved
9	FBB_EN	FRW	0H	Fast Back-to-Back Enable 1 = EPC will perform fast back-to-back transfers when bus master 0 = EPC will not perform fast back-to-back transfers
8	SERR_EN	FRW	0H	System Error Enable 1 = System error enabled: If PAR_EN (bit 6) is also enabled then $\overline{\text{SERR}}$ is driven in response to an address parity error. 0 = System error disabled: $\overline{\text{SERR}}$ is not driven.
7	-	R	0H	reserved
6	PAR_EN	FRW	0H	Parity Error Enable: 1 = EPC will report PCI parity errors 0 = EPC will ignore PCI parity errors
5	-	R	0H	reserved
4	-	R	0H	reserved

PCI_CMD (cont'd)

Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***

3	-	R	0H	reserved
2	MASTER_EN	FRW	0H	PCI Master Enable: 1 = EPC will act as PCI bus master (i.e. assert $\overline{\text{REQ}}$) 0 = EPC will not act as PCI bus master[a]
1	MEM_EN	FRW	0H	Memory Access Enable 1 = EPC will respond to memory accesses on the PCI bus 0 = EPC will ignore ALL memory accesses on the PCI bus
0	IO_EN	FRW	0H	I/O Access Enable 1 = EPC will respond to IO accesses on the PCI bus 0 = EPC will ignore ALL IO accesses on the PCI bus

[a] Clearing this bit effectively prohibits any local bus reads/writes to PCI space. If PCI bus mastering is disabled, all local bus writes to PCI space, and all DMA transfers destined for PCI space, will be queued in the Local-to-PCI FIFO until either this bit is set, or the FIFO is full.

PCI_STAT: PCI STATUS REGISTER

Mnemonic:	PCI_STAT
Offset:	06H
Size:	16 bits

PCI_STAT				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15	PAR_ERR	FRW	0H	Parity Error: set (1) in response to a parity error being detected on the PCI bus. Cleared by writing ’1’ to this bit.
14	SYS_ERR	FRW	0H	System Error: set (1) in response to a system error being detected by this device and reported on the $\overline{\text{SERR}}$ pin on the PCI bus. Cleared by writing ’1’ to this bit.
13	M_ABORT	FRW	0H	Master Abort: set (1) in response to a master abort being detected during transaction in which the EPC was acting as a bus master. Cleared by writing a '1' to this bit.
12	T_ABORT	FRW	0H	Target Abort: set (1) in response to a target abort being detected during transaction in which the EPC was acting as a bus master. Cleared by writing '1' to this bit.
11	-	R	0H	reserved
10-9	DEVSEL	FR	0H	Device Select Timing: Programmable during initialization for the benefit of other PCI bus masters. Doesn't affect the operation of the EPC.
8	PAR_REP	FRW	0H	Data Parity Error Report: set (1) whenever the EPC acts as a bus master and observes the $\overline{\text{PERR}}$ signal being driven. The PAR_EN bit in PCI_CMD must also be enabled for this bit to be set. Cleared by writing '1' to the bit.
7	FAST_BACK	FR	0H	Fast Back-to-Back Target Enable: Used to indicate to other bus masters the abillity of this device to respond to fast back-to-back transfers. Note: The state of this bit will not effect the internal operation of the EPC and it will always respond properly to fast back-to-back transfers.
6-0	-	R	0H	reserved

PCI_CC_REV: PCI CLASS AND REVISION REGISTER (PCI REQUIRED)

Mnemonic:	PCI_CC_REV
Offset:	08h
Size:	32 bits

PCI_CC_REV				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
31-24	BASE_CLASS	FR	0H[a]	PCI Base Class Code (see PCI specification)
23-16	SUB_CLASS	FR	0H	PCI Sub Class Code (see PCI specification)
15-8	PROG_IF	FR	0H	PCI Programming Interface Code (see PCI specification)
7-4	UREV	FR	0H	User Revision ID. These bits are programmable to indicate the revision level of the system built with the EPC.
3-0	VREV	R	Stepping ID	V3 Revision. These bits are hardwired to reflect the revision number of the component. Rev A=0h, Rev B0=1h, Rev B1=2h, Rev B2=3h

[a] 6H is the base class for a host to PCI bridge. This can be programmed to suit the application.

PCI_HDR_CFG: PCI HEADER/CONFIG. REGISTER (PCI REQUIRED)

Mnemonic:	PCI_HDR_CFG
Offset:	0CH
Size;	32 bits

PCI_HDR_CFG				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
31-24	BIST	R	0H	Built in Self Test: Unimplemented; reads back as all "0"
23-16	HDR_TYPE	R	0H	Header Type: Unimplemented; reads back as all "0"
15-11	LT	FRW	0H	Latency Timer: Latency value in multiples of 8 clocks.
10-8	LTL	R	0H	Latency Timer: Unimplemented lower bits of the PCI latency timer register
7-0	LINE_SIZE	FRW	0H	Cache Line Size: Has no effect on the internal operation of the EPC

PCI_I2O_BASE : PCI I_2O BASE ADDRESS REGISTER[a]

Mnemonic: PCI_I2O_BASE
Offset: 10H
Size: 32 bits

PCI_I2O_BASE				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
31-20	ADR_BASE	FRW	0H	Base Address: If the value of ADR_BASE matches that of AD[31:20] during the address phase of a PCI access then a match is detected. Since bits 31-20 are significant to the decoder, the size of the aperture is 1MB. This size can be increased using the corresponding ADR_SIZE register bits so that lower bits of the decode are masked off.
19-4	-	R	0H	reserved
3	PREFETCH	FR	0H	Prefetchable: enables prefetching of data for read cycles by anticipating sequential reads.
2-1	TYPE = "00"	R	0H	Address Range Type: These read only bits are hardwired to "00" to indicate that the device can be mapped anywhere in the 32 bit address space.
0	IO	FR	0H	‘1’ = access through I/O space. ‘0’ = access from memory space.

PCI_IO_BASE: PCI ACCESS TO INTERNAL EPC REGISTERS

Mnemonic: PCI_IO_BASE
Offset: 10H
Size: 32 bits

PCI_IO_BASE				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
31-8	ADR_BASE	FRW	0H	Base Address: If the value of ADR_BASE matches that of AD[31:8] during the address phase of a PCI access then a match is detected. Since bits 31-8 are significant to the decoder, the size of the PCI-to-Internal register aperture is 256 bytes.
7-4	-	R	0H	reserved (Zero)
3	PREFETCH	R	0H	Prefetchable. This bit is for configuration information only and has no affect on the operation of the EPC. Accesses to internal registers are never prefetched.
2-1	TYPE	R	0H	Address Range Type: These read only bits are hardwired to "00" to indicate that the device can be mapped anywhere in the 32 bit address space.
0	IO	FR	0H	1 = The PCI-to-Internal register aperture is in PCI I/O space 0 = The PCI-to-Internal register aperture is in PCI memory space

PCI_BASE0: PCI TO LOCAL BUS APERTURE 0 BASE ADDRESS

Mnemonic:	PCI_BASE0
Offset:	14H
Size:	32 bits

PCI_BASE0

Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
31-20	ADR_BASE	FRW	0H	Base Address: If the value of ADR_BASE matches that of AD[31:20] during the address phase of a PCI access then a match is detected. A larger address space can be decoded by changing the ADR_SIZE field in the PCI_MAP0 register. This will mask off some of the lower bits of this field to allow automatic configuration software to determine the size of the aperture.
19-8	ADR_BASEL	FRW	0H	Low order base address bits used for fine grain I/O decode only. These bits are only used when IO=1 and ADR_SIZE is set to 0100-0111 in the PCI_MAP0 register. (Not in Rev A)
7-4	-	R	0H	reserved
3	PREFETCH	FR	0H	Prefetchable: 1 = Enable read prefetching for this aperture 0 = Disable read prefetching for this aperture When LOCK is disabled and both the IO and PREFETCH bits are written to ’1’, the PREFETCH bit will read ’0’ even though it is internally set to ’1’ and the aperture will exhibit prefetch behavior.
2-1	TYPE	R	0H	Address Range Type: These read only bits are hardwired to "00" to indicate that the device can be mapped anywhere in the 32 bit address space.
0	IO	FR	0H	1 = The PCI-to-Local aperture 0 will respond to PCI IO space access ($\overline{\text{CBE}}$=2h, 3h) 0 = The PCI-to-Local aperture 0 will respond to PCI memory space access ($\overline{\text{CBE}}$=6h, 7h, Ch, Eh, Fh)

PCI_BASE1: PCI TO LOCAL BUS APERTURE 1 BASE ADDRESS

Mnemonic:	PCI_BASE1
Offset:	18H
Size:	32 bits

PCI_BASE1				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
31-20	ADR_BASE	FRW	0H	Base Address: If the value of ADR_BASE matches that of AD[31:20] during the address phase of a PCI access then a match is detected. In legacy DOS mode the address comparison has increased granularity (see DOS Compatibility chapter).
19-14	ADR_BASEL (Not in Rev A)	FRW R	0H	Base address bits used only for DOS compatibility mode. See PC Compatibility chapter. These bits read back as '0' unless DOS mode is selected in the PCI_MAP1 register.
13-11	-	R	0H	reserved
10-8	DOS_MEM (Not in Rev A)	FRW R	0H	DOS Mode Memory Size: When IO=0 and DOS mode is selected, these bits set the size of the real mode DOS memory hole: 100 = 16K bytes (A[31:14]) 101 = 32K bytes (A[31:15]) 110 = 64K bytes (A[31:16]) 111 = 128K bytes (A[31:17]) others = disabled These bits read back as '0' unless DOS mode is selected in the PCI_MAP1 register.
7-4	-	R	0H	reserved
3	PREFETCH	FR	0H	Prefetchable: 1 = Enable read prefetching for this aperture 0 = Disable read prefetching for this aperture When LOCK is disabled and both the IO and PREFETCH bits are written to '1', the PREFETCH bit will read '0' even though it is internally set to '1' and the aperture will exhibit prefetch behavior.
2-1	TYPE	R	0H	Address Range Type: These read only bits are hardwired to "00" to indicate that the device can be mapped anywhere in the 32 bit address space.
0	IO	FR	0H	1 = The PCI-to-Local aperture 0 is in PCI IO space 0 = The PCI-to-Local aperture 0 is in PCI memory space

PCI_SUB_VENDOR: PCI SUBSYSTEM VENDOR

Mnemonic:	PCI_SUB_VENDOR
Offset:	2CH
Size:	16 bits

PCI_SUB_VENDOR				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15-0	VENDOR	FRW	0H	Subsystem Vendor ID: firmware programmable (PCI 2.1 and Windows95® required) (not in Rev A)

PCI_SUB_VENDOR: PCI SUBSYSTEM ID

Mnemonic:	PCI_SUB_ID
Offset:	2EH
Size:	16 bits

PCI_SUB_ID

Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15-0	ID	FRW	0H	Subsystem ID: firmware programmable (PCI 2.1 and Windows95® required) (not in Rev A)

PCI_ROM: EXPANSION ROM BASE

Mnemonic:	PCI_ROM
Offset:	30H
Size:	32 bits

PCI_ROM

Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
31-12	ROM_BASE	FRW	0H	Expansion ROM Base Address. The size of the ROM aperture is controlled in PCI_MAP0 and the decoders are shared between these apertures. (Not in Rev A)
11-1	-	R	0H	reserved
0	ENABLE	FRW	0H	Expansion ROM Enable. 1 = enabled, 0 = disabled (Not in Rev A)

PCI_BPARAM: PCI BUS PARAMETER REGISTER (PCI REQUIRED)

Mnemonic:	PCI_BPARAM
Offset:	3CH
Size:	32 bits

PCI_BPARAM

Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
31-24	MAX_LAT	FR	0H	Maximum Latency: For PCI autoconfiguration reporting only. Has no effect on the internal operation of the EPC
23-16	MIN_GNT	FR	0H	Minimum Grant: For PCI autoconfiguration reporting only. Has no effect on the internal operation of the EPC
15-11	-	R	0H	reserved
10-8	INT_PIN	FRW	0H	Interrupt Pin: Selects which interrupt pin will be driven to the PCI bus (PCI Spec). Chooses which INTx pin will receive internal EPC interrupts (see "Interrupt Control" chapter). 000 = Disabled 001 = Use INTA 010 = Use INTB 011 = Use INTC 100 = Use INTD
7-0	INT_LINE	FRW	0H	Interrupt Line: For PCI autoconfiguration reporting only. Has no effect on the internal operation of the EPC

PCI_MAP0: PCI BUS TO LOCAL BUS ADDRESS MAP 0

Mnemonic:	PCI_MAP0
Offset:	40H
Size:	32 bits

.

<table>
<tr><th colspan="5">PCI_MAP0</th></tr>
<tr><th><i>Bits</i></th><th><i>Mnemonic</i></th><th><i>Type</i></th><th><i>Reset Value</i></th><th><i>Description</i></th></tr>
<tr><td>31-20</td><td>MAP_ADR</td><td>FRW</td><td>0H</td><td>Map Address: These bits correspond to bits LAD[31:20] in local address space when a PCI to Local access is made. The lower bits of MAP_ADR are masked off according to the ADR_SIZE bits in the PCI_MAP registers.</td></tr>
<tr><td>19-16</td><td>-</td><td>R</td><td>0H</td><td>reserved</td></tr>
<tr><td>15</td><td>RD_POST_INH
(Not in Rev A)</td><td>FRW</td><td>0H</td><td>Read Posting Inhibit: When set ’1’ the very first read of a burst read from the corresponding aperture will not generate a STOP regardless of the latency of the access.</td></tr>
<tr><td>14-12</td><td>-</td><td>R</td><td>0H</td><td>reserved</td></tr>
<tr><td>11-10</td><td>ROM_SIZE</td><td>FRW</td><td>0H</td><td>ROM Size: Determines the size of the expansion ROM address decoder (Not in Rev A):
00 = expansion ROM base register disabled
01 = 4K byte expansion ROM (A[31:12] significant)
10 = 16K byte expansion ROM (A[31:14] significant)
11 = 64K byte expansion ROM (A[31:16] significant)</td></tr>
<tr><td>9-8</td><td>SWAP</td><td>FRW</td><td>0H</td><td>Byte Swap Control: Selects byte lane swapping for read and write cycles according to the following table:
<table>
<tr><th></th><th>SWAP</th><th>D[31:24]</th><th>D[23:26]</th><th>D[15:8]</th><th>D[7:0]</th></tr>
<tr><td>no swap, 32 bit</td><td>00</td><td>Q[31:24]</td><td>Q[23:16]</td><td>Q[15:8]</td><td>Q[7:0]</td></tr>
<tr><td>16 bit</td><td>01</td><td>Q[15:8]</td><td>Q[7:0]</td><td>Q[31:24]</td><td>Q[23:16]</td></tr>
<tr><td>8 bit</td><td>10</td><td>Q[7:0]</td><td>Q[15:8]</td><td>Q[23:16]</td><td>Q[31:24]</td></tr>
<tr><td></td><td>11</td><td colspan="4">Auto Swap (Rev B2 only). Reserved in B0/B1</td></tr>
</table>
Auto Swap: When local bus BE[3:0] = “1100” or “0011” then a 16 bit swap is done. When local bus BE[3:0] = “1110”, “1101”, “1011” or “0111” then an 8 bit swap is done. Any other combination results in non-swapped data.</td></tr>
</table>

<table>
<tr><th colspan="5">PCI_MAP0 (cont'd)</th></tr>
<tr><th>Bits</th><th>Mnemonic</th><th>Type</th><th>Reset Value</th><th>Description</th></tr>
<tr><td>7-4</td><td>ADR_SIZE</td><td>FRW</td><td>0H</td><td>Aperture Size: Legacy DOS mode uses a different decoding scheme described in the "PC Compatibility" chapter of the manual.
<table>
<tr><th>ADDR_SIZE</th><th>Size</th><th>Valid ADR BASE Bits</th></tr>
<tr><td>0000</td><td>1MB</td><td>31:20</td></tr>
<tr><td>0001</td><td>2MB</td><td>31:21</td></tr>
<tr><td>0010</td><td>4MB</td><td>31:22</td></tr>
<tr><td>0011</td><td>8MB</td><td>31:23</td></tr>
<tr><td>0100</td><td>16MB memory
256 byte I/O</td><td>31:24 (mem)
31:8 (IO)</td></tr>
<tr><td>0101</td><td>32MB memory
512 byte I/O</td><td>31:25 (mem)
31:9 (IO)</td></tr>
<tr><td>0110</td><td>64MB memory
1024 byte I/O</td><td>31:26 (mem)
31:10 (IO)</td></tr>
<tr><td>0111</td><td>128MB memory
2048 byte I/O</td><td>31:27 (mem)
31:11 (IO)</td></tr>
<tr><td>1000</td><td>256MB</td><td>31:28</td></tr>
<tr><td>1001[a]</td><td>512MB</td><td>31:29</td></tr>
<tr><td>1010[a]</td><td>1GB</td><td>31:30</td></tr>
<tr><td>11xx[a]</td><td>1MB DOS Mode
(PCI_MAP1 only)</td><td>31:20</td></tr>
<tr><td>others</td><td>-</td><td>reserved</td></tr>
</table>
[a] (Revision B2 only. Reserved in Revision B1/B0.</td></tr>
<tr><td>3-2</td><td>-</td><td>R</td><td>0H</td><td>reserved</td></tr>
<tr><td>1</td><td>REG_EN</td><td>FRW</td><td>0H</td><td>PCI_BASE0 register enable. 1 = PCI_BASE0 enabled, 0=PCI_BASE0 disabled (reads back as 0H). (Not in Rev A)</td></tr>
<tr><td>0</td><td>ENABLE</td><td>FRW</td><td>0H</td><td>PCI_BASE0 Aperture Enable. 1 = PCI-to-Local aperture 0 is enabled, 0 = PCI-to-Local aperture 0 is disabled. (Not in Rev A)</td></tr>
</table>

PCI_MAP1: PCI BUS TO LOCAL BUS ADDRESS MAP 1

Mnemonic: PCI_MAP1
Offset: 44H
Size: 32 bits
.

<table>
<tr><th colspan="5">PCI_MAP1</th></tr>
<tr><th><em>Bits</em></th><th><em>Mnemonic</em></th><th><em>Type</em></th><th><em>Reset Value</em></th><th><em>Description</em></th></tr>
<tr><td>31-20</td><td>MAP_ADR</td><td>FRW</td><td>0H</td><td>Map Address: These bits correspond to bits LAD(31:20) in local address space when a PCI to Local access is made. The lower bits of MAP_ADR are masked off according to the ADR_SIZE bits in the PCI_MAP registers.</td></tr>
<tr><td>19-16</td><td>-</td><td>R</td><td>0H</td><td>reserved</td></tr>
<tr><td>15</td><td>RD_POST_INH
(Not in Rev A)</td><td>FRW</td><td>0H</td><td>Read Posting Inhibit: When set ’1’ the very first read of a burst read from the corresponding aperture will not generate a $\overline{\text{STOP}}$ regardless of the latency of the access.</td></tr>
<tr><td>14-10</td><td>-</td><td>R</td><td>0H</td><td>reserved</td></tr>
<tr><td>9-8</td><td>SWAP</td><td>FRW</td><td>0H</td><td>Byte Swap Control: Selects byte lane swapping for read and write cycles according to the following table:
<table>
<tr><th></th><th>SWAP</th><th>D[31:24]</th><th>D[23:26]</th><th>D[15:8]</th><th>D[7:0]</th></tr>
<tr><td>no swap, 32 bit</td><td>00</td><td>Q[31:24]</td><td>Q[23:16]</td><td>Q[15:8]</td><td>Q[7:0]</td></tr>
<tr><td>16 bit</td><td>01</td><td>Q[15:8]</td><td>Q[7:0]</td><td>Q[31:24]</td><td>Q[23:16]</td></tr>
<tr><td>8 bit</td><td>10</td><td>Q[7:0]</td><td>Q[15:8]</td><td>Q[23:16]</td><td>Q[31:24]</td></tr>
<tr><td></td><td>11</td><td colspan="4">Auto Swap (Rev B2 only). Reserved in B0/B1</td></tr>
</table>
Auto Swap: When local bus $\overline{\text{BE[3:0]}}$ = “1100” or “0011” then a 16 bit swap is done. When local bus $\overline{\text{BE[3:0]}}$ = “1110”, “1101”, “1011” or “0111” then an 8 bit swap is done. Any other combination results in non-swapped data.</td></tr>
</table>

<table>
<tr><th colspan="5">PCI_MAP1 (Cont’d)</th></tr>
<tr><th><em>Bits</em></th><th><em>Mnemonic</em></th><th><em>Type</em></th><th><em>Reset Value</em></th><th><em>Description</em></th></tr>
<tr><td>7-4</td><td>ADR_SIZE</td><td>FRW</td><td>0H</td><td>Aperture Size: Legacy DOS mode uses a different decoding scheme described in the "PC Compatibility" chapter of the manual.
<table>
<tr><th>ADDR_SIZE</th><th>Size</th><th>Valid ADR BASE Bits</th></tr>
<tr><td>0000</td><td>1MB</td><td>31:20</td></tr>
<tr><td>0001</td><td>2MB</td><td>31:21</td></tr>
<tr><td>0010</td><td>4MB</td><td>31:22</td></tr>
<tr><td>0011</td><td>8MB</td><td>31:23</td></tr>
<tr><td>0100</td><td>16MB memory
256 byte I/O</td><td>31:24 (mem)
31:8 (IO)</td></tr>
<tr><td>0101</td><td>32MB memory
512 byte I/O</td><td>31:25 (mem)
31:9 (IO)</td></tr>
<tr><td>0110</td><td>64MB memory
1024 byte I/O</td><td>31:26 (mem)
31:10 (IO)</td></tr>
<tr><td>0111</td><td>128MB memory
2048 byte I/O</td><td>31:27 (mem)
31:11 (IO)</td></tr>
<tr><td>1000</td><td>256MB</td><td>31:28</td></tr>
<tr><td>1001</td><td>512MB</td><td>31:29</td></tr>
<tr><td>1010</td><td>1GB</td><td>31:30</td></tr>
<tr><td>11xx</td><td>1MB DOS Mode
(PCI_MAP1 only)</td><td>31:20</td></tr>
<tr><td>others</td><td>-</td><td>reserved</td></tr>
</table>
</td></tr>
<tr><td>3-2</td><td>-</td><td>R</td><td>0H</td><td>reserved</td></tr>
<tr><td>1</td><td>REG_EN</td><td>FRW</td><td>0H</td><td>PCI_BASE1 register enable. 1 = PCI_BASE1 enabled, 0=PCI_BASE1 disabled (reads back as 0H from PCI and Local). (Not in Rev A)</td></tr>
<tr><td>0</td><td>ENABLE</td><td>FRW</td><td>0H</td><td>PCI_BASE1 Aperture Enable. 1 = PCI-to-Local aperture 1 is enabled, 0 = PCI-to-Local aperture 1 is disabled. (Not in Rev A)</td></tr>
</table>

PCI_I2O_MAP: PCI BUS I$_2$O ATU LOCAL BUS ADDRESS MAP

Mnemonic:	PCI_I2O_MAP
Offset:	44H
Size:	32 bits

<table>
<tr><th colspan="5">PCI_I2O_MAP</th></tr>
<tr><th>Bits</th><th>Mnemonic</th><th>Type</th><th>Reset Value</th><th>Description</th></tr>
<tr><td>31-20</td><td>MAP_ADR</td><td>FRW</td><td>0H</td><td>Map Address: These bits correspond to bits LAD(31:20) in local address space when a PCI to Local access is made. Address bits LAD(19:2) are derived from the PCI bus itself (for a 1MB aperture size). If the size of the aperture is increased, then the lower bits of MAP_ADR become masked off according to the ADR_SIZE bits in the PCI_MAP registers.</td></tr>
<tr><td>19-16</td><td>-</td><td>R</td><td>0H</td><td>reserved</td></tr>
<tr><td>15</td><td>RD_POST_DIS (Not in Rev A</td><td>FR</td><td>0H</td><td>Read Post Disable: When set '1' the very first read of a burst read from the corresponding aperture will not generate a STOP#.</td></tr>
<tr><td>14-10</td><td>-</td><td>R</td><td>0H</td><td>reserved</td></tr>
<tr><td>9-8</td><td>SWAP</td><td>FRW</td><td>0H</td><td>Byte Swap Control: Selects byte lane swapping for read and write cycles according to the following table:
<table>
<tr><th></th><th>SWAP</th><th>D[31:24]</th><th>D[23:26]</th><th>D[15:8]</th><th>D[7:0]</th></tr>
<tr><td>no swap, 32 bit</td><td>00</td><td>Q[31:24]</td><td>Q[23:16]</td><td>Q[15:8]</td><td>Q[7:0]</td></tr>
<tr><td>16 bit</td><td>01</td><td>Q[15:8]</td><td>Q[7:0]</td><td>Q[31:24]</td><td>Q[23:16]</td></tr>
<tr><td>8 bit</td><td>10</td><td>Q[7:0]</td><td>Q[15:8]</td><td>Q[23:16]</td><td>Q[31:24]</td></tr>
<tr><td></td><td>11</td><td colspan="4">Auto Swap (Rev B2 only). Reserved in B0/B1</td></tr>
</table>
Auto Swap: When local bus $\overline{\text{BE[3:0]}}$ = “1100” or “0011” then a 16 bit swap is done. When local bus $\overline{\text{BE[3:0]}}$ = “1110”, “1101”, “1011” or “0111” then an 8 bit swap is done. Any other combination results in non-swapped data.</td></tr>
<tr><td>7-4</td><td>ADR_SIZE</td><td>FRW</td><td>0H</td><td>Aperture Size: Legacy DOS mode uses a different decoding scheme described in the "PC Compatibility" chapter of the manual.
<table>
<tr><th>ADDR_SIZE</th><th>Size</th><th>Valid ADR BASE Bits</th></tr>
<tr><td>0000</td><td>1MB</td><td>31:20</td></tr>
<tr><td>0001</td><td>2MB</td><td>31:21</td></tr>
<tr><td>0010</td><td>4MB</td><td>31:22</td></tr>
<tr><td>0011</td><td>8MB</td><td>31:23</td></tr>
<tr><td>0100</td><td>16MB memory
256 byte I/O</td><td>31:24 (mem)
31:8 (IO)</td></tr>
<tr><td>0101</td><td>32MB memory
512 byte I/O</td><td>31:25 (mem)
31:9 (IO)</td></tr>
<tr><td>0110</td><td>64MB memory
1024 byte I/O</td><td>31:26 (mem)
31:10 (IO)</td></tr>
<tr><td>0111</td><td>128MB memory
2048 byte I/O</td><td>31:27 (mem)
31:11 (IO)</td></tr>
<tr><td>1000</td><td>256MB</td><td>31:28</td></tr>
<tr><td>1001</td><td>512MB</td><td>31:29</td></tr>
<tr><td>1010</td><td>1GB</td><td>31:30</td></tr>
<tr><td>11xx</td><td>1MB DOS Mode (PCI_MAP1 only)</td><td>31:20</td></tr>
<tr><td>others</td><td>-</td><td>reserved</td></tr>
</table></td></tr>
<tr><td>3-2</td><td>-</td><td>R</td><td>0H</td><td>reserved</td></tr>
</table>

PCI_I2O_MAP (cont’d)				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
1	REG_EN	FRW	0H	PCI_BASE1 register enable. 1 = PCI_BASE1 enabled, 0=PCI_BASE1 disabled (reads back as 0H from PCI and Local). (Not in Rev A)
0	ENABLE	FRW	0H	PCI_BASE1 Aperture Enable. 1 = PCI-to-Local aperture 1 is enabled, 0 = PCI-to-Local aperture 1 is disabled. (Not in Rev A)

PCI_INT_STAT: PCI INTERRUPT STATUS REGISTER

Mnemonic:	PCI_INT_STAT
Offset:	48H
Size:	32 bits

PCI_INT_STAT				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
31	MAILBOX	R	0H	Mailbox Interrupt: 1 = Mailbox (Doorbell) Interrupt request active 0 = No mailbox interrupts pending Cleared by clearing MAIL_RD_STAT and MAIL_WR_STAT
30	LOCAL	FRW	0H	Local Bus Direct Interrupt: 1 = Local bus master requests a PCI interrupt 0 = No operation This bit is set by writing ’1’ and cleared by writing ‘0’
29-28	-	R	0H	reserved
27	OUT_POST	FRW	0H	I2O Outbound Post List Not Empty: Indicates that the outbound post list head pointer not equals the tail pointer (OPL_HEAD^{1}OPL_TAIL). This bit is equivalent to the PCI_I2O_ISTAT register bit 3 and can be read there also. It is masked off only when the I2O_EN bit in the PCI_CFG register is clear otherwise the not empty status will be readable here regardless of the mask bit in PCI_INT_CFG. This bit is also mapped into the PCI_I2O_ISTAT register bit 3 and can be read there also.
26	-	R	0H	reserved
25	DMA1	FRW	0H	DMA channel 1 Interrupt: 1 = DMA channel 1 has requested an interrupt 0 = DMA channel 1 has not requested an interrupt
24	DMA0	FRW	0H	DMA channel 0 Interrupt: 1 = DMA channel 0 has requested an interrupt 0 = DMA channel 0 has not requested an interrupt
23-15		R	0H	reserved
14	INTC_TO_D	FRW	0H	$\overline{\text{INTD}}$ Output from $\overline{\text{INTC}}$ Input: Set (’1’) when enabled (INTC_INTD bit in the corresponding PCI_INT_CFG register field) *and* $\overline{\text{INTC}}$ is used as an input *and* an interrupt event has occurred on $\overline{\text{INTC}}$
13	INTB_TO_D	FRW	0H	$\overline{\text{INTD}}$ Output from $\overline{\text{INTB}}$ Input: Set (’1’) when enabled (INTB_INTD bit in the corresponding PCI_INT_CFG register field) *and* $\overline{\text{INTB}}$ is used as an input *and* an interrupt event has occurred on $\overline{\text{INTB}}$

PCI_INT_STAT (cont’d)				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
12	INTA_TO_D	FRW	0H	INTD Output from INTA Input: Set (’1’) when enabled (INTA_INTD bit in the corresponding PCI_INT_CFG register field) *and* INTA is used as an input *and* an interrupt event has occurred on INTA
11	INTD_TO_C	FRW	0H	See description above for INTx_TO_y[a]
10	-	R	0H	reserved
9	INTB_TO_C	FRW	0H	See description above for INTx_TO_y
8	INTA_TO_C	FRW	0H	See description above for INTx_TO_y
7	INTD_TO_B	FRW	0H	See description above for INTx_TO_y
6	INTC_TO_B	FRW	0H	See description above for INTx_TO_y
5	-	R	0H	reserved
4	INTA_TO_B	FRW	0H	See description above for INTx_TO_y
3	INTD_TO_A	FRW	0H	See description above for INTx_TO_y
2	INTC_TO_A	FRW	0H	See description above for INTx_TO_y
1	INTB_TO_A	FRW	0H	See description above for INTx_TO_y
0	-	R	0H	reserved

[a] All of the INTx_TO_y bits function identically with "x" being the source of the interrupt (PCI INTx) and "y" being the destination for the request (PCI INTy).

Note: LOCAL interrupt request is cleared by writing "0"; writing ’1’ has no effect.
All other writable status bits are cleared by writing ’1’; writing ’0’ has no effect.

PCI_INT_CFG: PCI INTERRUPT CONFIGURATION REGISTER

Mnemonic:	PCI_INT_CFG
Offset:	4CH
Size:	32 bits

PCI_INT_CFG				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
31	MAILBOX	FRW	0H	Mailbox Interrupt Enable: Enables a PCI interrupt from the mailbox unit. (see the Mailbox Registers chapter).
30	LOCAL	FRW	0H	Local Bus Direct Interrupt Enable: Enables direct local bus to PCI interrupts
29	MASTER_PI	FRW	0H	PCI Master Local Interrupt Enable: When enabled (1) together with the PCI_PERR bit in LB_IMASK (bit 3), a local bus interrupt will be generated whenever the VxxxEPC acts as a bus master and a parity error occurs.
28	SLAVE_PI	FRW	0H	PCI Slave Local Interrupt Enable: When enabled (1) together with the PCI_PERR bit in LB_IMASK (bit 3), a local bus interrupt will be generated whenever the VxxxEPC acts as a bus slave and a parity error occurs.
27	OUT_POST	R	0H	I2O Outbound Post List Not Empty: When Enabled (‘1’) the PCI interrupt pin (selected by the INT_PIN field of the PCI_BPARAM register) is asserted whenever the outbound post list head pointer not equals the tail pointer (OPL_HEAD¹OPL_TAIL). This bit is equivalent to the PCI_I2O_MASK register bit 3 and can be read/written there also.
26	-	R	0H	reserved

<table>
<tr><th colspan="5">PCI_INT_CFG (cont’d)</th></tr>
<tr><th>Bits</th><th>Mnemonic</th><th>Type</th><th>Reset Value</th><th>Description</th></tr>
<tr><td>25</td><td>DMA1</td><td>FRW</td><td>0H</td><td>DMA Channel 1 interrupt enable</td></tr>
<tr><td>24</td><td>DMA0</td><td>FRW</td><td>0H</td><td>DMA Channel 0 interrupt enable</td></tr>
<tr><td>23-22</td><td>MODE_D</td><td>FRW</td><td>0H</td><td>INTD Interrupt Mode: Determines use of the corresponding interrupt pin.
<table>
<tr><th>MODE_D</th><th>Description</th></tr>
<tr><td>00</td><td>Active low level triggered input</td></tr>
<tr><td>01</td><td>High-to-low edge triggered input</td></tr>
<tr><td>10</td><td>Software cleared output. INTD pin is asserted via an interrupt event and cleared through the PCI_INT_STAT register.</td></tr>
<tr><td>11</td><td>reserved</td></tr>
</table></td></tr>
<tr><td>21-20</td><td>MODE_C</td><td>FRW</td><td>0H</td><td>INTC Interrupt Mode
Note: See MODE_D description for bit settings.</td></tr>
<tr><td>19-18</td><td>MODE_B</td><td>FRW</td><td>0H</td><td>INTB Interrupt Mode
Note: See MODE_D description for bit settings.</td></tr>
<tr><td>17-16</td><td>MODE_A</td><td>FRW</td><td>0H</td><td>INTA Interrupt Mode
Note: See MODE_D description for bit settings.</td></tr>
<tr><td>15</td><td>INTD_TO_LB</td><td>FRW</td><td>0H</td><td>1 = INTD will request local ICU interrupts when the input is active
0 = INTD will never request LICU interrupts</td></tr>
<tr><td>14</td><td>INTC_TO_D</td><td>FRW</td><td>0H</td><td>1 = INTC will act as interrupt request for INTD output
0 = INTC will not act as interrupt request for INTD output</td></tr>
<tr><td>13</td><td>INTB_TO_D</td><td>FRW</td><td>0H</td><td>1 = INTB will act as interrupt request for INTD output
0 = INTB will not act as interrupt request for INTD output</td></tr>
<tr><td>12</td><td>INTA_TO_D</td><td>FRW</td><td>0H</td><td>1 = INTA will act as interrupt request for INTD output
0 = INTA will not act as interrupt request for INTD output</td></tr>
<tr><td>11</td><td>INTD_TO_C</td><td>FRW</td><td>0H</td><td>1 = INTD will act as interrupt request for INTC output
0 = INTD will not act as interrupt request for INTC output</td></tr>
<tr><td>10</td><td>INTC_TO_LB</td><td>FRW</td><td>0H</td><td>1 = INTC will request local ICU interrupts when the input is active
0 = INTC will never request LICU interrupts</td></tr>
<tr><td>9</td><td>INTB_TO_C</td><td>FRW</td><td>0H</td><td>1 = INTB will act as interrupt request for INTC output
0 = INTB will not act as interrupt request for INTC output</td></tr>
<tr><td>8</td><td>INTA_TO_C</td><td>FRW</td><td>0H</td><td>1 = INTA will act as interrupt request for INTC output
0 = INTA will not act as interrupt request for INTC output</td></tr>
<tr><td>7</td><td>INTD_TO_B</td><td>FRW</td><td>0H</td><td>1 = INTD will act as interrupt request for INTB output
0 = INTD will not act as interrupt request for INTB output</td></tr>
<tr><td>6</td><td>INTC_TO_B</td><td>FRW</td><td>0H</td><td>1 = INTC will act as interrupt request for INTB output
0 = INTC will not act as interrupt request for INTB output</td></tr>
<tr><td>5</td><td>INTB_TO_LB</td><td>FRW</td><td>0H</td><td>1 = INTB will request local ICU interrupts when the input is active
0 = INTB will never request LICU interrupts</td></tr>
<tr><td>4</td><td>INTA_TO_B</td><td>FRW</td><td>0H</td><td>1 = INTA will act as interrupt request for INTB output
0 = INTA will not act as interrupt request for INTB output</td></tr>
<tr><td>3</td><td>INTD_TO_A</td><td>FRW</td><td>0H</td><td>1 = INTD will act as interrupt request for INTA output
0 = INTD will not act as interrupt request for INTA output</td></tr>
<tr><td>2</td><td>INTC_TO_A</td><td>FRW</td><td>0H</td><td>1 = INTC will act as interrupt request for INTA output
0 = INTC will not act as interrupt request for INTA output</td></tr>
<tr><td>1</td><td>INTB_TO_A</td><td>FRW</td><td>0H</td><td>1 = INTB will act as interrupt request for INTA output
0 = INTB will not act as interrupt request for INTA output</td></tr>
<tr><td>0</td><td>INTA_TO_LB</td><td>FRW</td><td>0H</td><td>1 = INTA will request local ICU interrupts when the input is active
0 = INTA will never request LICU interrupts</td></tr>
</table>

LB_BASE 0,1 : LOCAL BUS TO PCI BUS APERTURE 0,1 ADDRESS

Mnemonic:	LB_BASE0, 1
Offset:	54H, 58H
Size:	32 bits

<table>
<tr><th colspan="5">LB_BASE0, 1</th></tr>
<tr><th>Bits</th><th>Mnemonic</th><th>Type</th><th>Reset Value</th><th>Description</th></tr>
<tr><td>31-20</td><td>ADR_BASE</td><td>FRW</td><td>0H</td><td>Base Address: If the value of ADR_BASE matches that of local address bits 31:20 during the address phase of a local access then a match is detected. Since bits 31-20 are significant to the decoder, the size of the aperture is 1MB. The aperture size can be increased using the corresponding ADR_SIZE register bits so that lower bits of the decode are masked off.</td></tr>
<tr><td>19-10</td><td>-</td><td>R</td><td>0H</td><td>reserved</td></tr>
<tr><td>9-8</td><td>SWAP</td><td>FRW</td><td>0H</td><td>Byte Swap Control: Selects byte lane swapping for read and write cycles according to the following table:
<table>
<tr><th></th><th>SWAP</th><th>D[31:24]</th><th>D[23:26]</th><th>D[15:8]</th><th>D[7:0]</th></tr>
<tr><td>no swap, 32 bit</td><td>00</td><td>Q[31:24]</td><td>Q[23:16]</td><td>Q[15:8]</td><td>Q[7:0]</td></tr>
<tr><td>16 bit</td><td>01</td><td>Q[15:8]</td><td>Q[7:0]</td><td>Q[31:24]</td><td>Q[23:16]</td></tr>
<tr><td>8 bit</td><td>10</td><td>Q[7:0]</td><td>Q[15:8]</td><td>Q[23:16]</td><td>Q[31:24]</td></tr>
<tr><td></td><td>11</td><td colspan="4">Auto Swap (Rev B2 only). Reserved in B0/B1</td></tr>
</table>
Auto Swap: When local bus $\overline{\mathrm{BE}[3:0]}$ = “1100” or “0011” then a 16 bit swap is done. When local bus $\overline{\mathrm{BE}[3:0]}$ = “1110”, “1101”, “1011” or “0111” then an 8 bit swap is done. Any other combination results in non-swapped data.</td></tr>
<tr><td>7-4</td><td>ADR_SIZE</td><td>FRW</td><td>0H</td><td>Aperture Size: The size of the aperture is determined as follows:
<table>
<tr><th>ADDR_SIZE</th><th>Size</th><th>Valid ADR BASE Bits</th></tr>
<tr><td>0000</td><td>1MB</td><td>31:20</td></tr>
<tr><td>0001</td><td>2MB</td><td>31:21</td></tr>
<tr><td>0010</td><td>4MB</td><td>31:22</td></tr>
<tr><td>0011</td><td>8MB</td><td>31:23</td></tr>
<tr><td>0100</td><td>16MB</td><td>31:24</td></tr>
<tr><td>0101</td><td>32MB</td><td>31:25</td></tr>
<tr><td>0110</td><td>64MB</td><td>31:26</td></tr>
<tr><td>0111</td><td>128MB</td><td>31:27</td></tr>
<tr><td>1000</td><td>256MB</td><td>31:28</td></tr>
<tr><td>1001[a]</td><td>512MB</td><td>31:29</td></tr>
<tr><td>1010[b]</td><td>1GB</td><td>31:29</td></tr>
<tr><td>1011[b]</td><td>2GB</td><td>31:29</td></tr>
<tr><td>others</td><td colspan="2">reserved</td></tr>
</table>
a.(Revision B2 only. Reserved in Revision B1/B0.
b.(Revision B2 only. Reserved in Revision B1/B0. The 1GB and 2GB aperture can be placed on a 512MB boundary</td></tr>
<tr><td>3</td><td>PREFETCH</td><td>FRW</td><td>0H</td><td>Prefetch Enable:
1 = enable the aperture for read prefetching
0 = disable read prefetching</td></tr>
<tr><td>2-1</td><td>-</td><td>R</td><td>0H</td><td>reserved</td></tr>
<tr><td>0</td><td>ENABLE</td><td>FRW</td><td>0H</td><td>1 = enable Local-to-PCI aperture 0.
0 = disable Local-to-PCI aperture 0.</td></tr>
</table>

LB_MAP0, 1: LOCAL BUS TO PCI BUS ADDRESS MAP 0, 1

Mnemonic:	LB_MAP0,1
Offset:	5EH, 62H
Size:	16 bits

LB_MAP0, 1				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15-4	MAP_ADR	FRW	0H	Map Address: These bits correspond to bits AD[31:20] in PCI address space when a Local to PCI access is made. Address bits AD[19:2] are derived from the local bus itself. If the size of the aperture is increased, then the lower bits of MAP_ADR become masked off according to the ADR_SIZE bits in the LB_BASE registers.
3-1	TYPE	FRW	0H	Access Type: Determines which PCI bus command will be driven for local bus to PCI bus access:[a] 000 = Interrupt Acknowledge (Read) 001 = I/O Read/Write 011 = Memory Read/Write 101 = Configuration Read/Write 110 = Memory Read Multiple/Memory Write others = reserved
0	AD_LOW_EN	FRW	0H	Low Address Override Enable: When Set (1) the AD[1:0] value transmitted during the address phase will be generated from the AD_LOW value in the PCI_CFG register. When cleared (0) the value of AD[1:0] will be “00” except for I/O cycles where AD[1:0] correspond to the byte enables.

[a] The value in this bit field is driven on the C/BE[3:1] PCI bus pins directly (except for TYPE="110": see note). C/BE0 is set based on whether the cycle is a read (0) or a write (1). No checking is done by the EPC device to see if the command type is supported (see the "Aperture Operation" section for more details). Reserved combinations can be used to generate other PCI commands directly.

LB_BASE2: LOCAL BUS TO PCI BUS I/O APERTURE ADDRESS[a]

Mnemonic: LB_BASE2
Offset: 64H
Size: 16 bits

<table>
<tr><th colspan="5">LB_BASE 2</th></tr>
<tr><th><i>Bits</i></th><th><i>Mnemonic</i></th><th><i>Type</i></th><th><i>Reset Value</i></th><th><i>Description</i></th></tr>
<tr><td>15-8</td><td>ADR_BASE</td><td>FRW</td><td>0H</td><td>Base Address: If the value of ADR_BASE matches that of local address bits 31:24 during the address phase of a local access then a match is detected. Since bits 31-24 are significant to the decoder, the size of the aperture is 16MB and is fixed at that size.</td></tr>
<tr><td>7-6</td><td>SWAP</td><td>FRW</td><td>0H</td><td>Byte Swap Control: Selects byte lane swapping for read and write cycles according to the following table:
<table>
<tr><th></th><th>SWAP</th><th>D[31:24]</th><th>D[23:26]</th><th>D[15:8]</th><th>D[7:0]</th></tr>
<tr><td>no swap, 32 bit</td><td>00</td><td>Q[31:24]</td><td>Q[23:16]</td><td>Q[15:8]</td><td>Q[7:0]</td></tr>
<tr><td>16 bit</td><td>01</td><td>Q[15:8]</td><td>Q[7:0]</td><td>Q[31:24]</td><td>Q[23:16]</td></tr>
<tr><td>8 bit</td><td>10</td><td>Q[7:0]</td><td>Q[15:8]</td><td>Q[23:16]</td><td>Q[31:24]</td></tr>
<tr><td></td><td>11</td><td colspan="4">Auto Swap (Rev B2 only). Reserved in B0/B1</td></tr>
</table>
Auto Swap: When local bus $\overline{BE[3:0]}$ = “1100” or “0011” then a 16 bit swap is done. When local bus $\overline{BE[3:0]}$ = “1110”, “1101”, “1011” or “0111” then an 8-bit swap is done. Any other combination results in non-swapped data.</td></tr>
<tr><td>5-1</td><td>-</td><td>R</td><td>0H</td><td>reserved</td></tr>
<tr><td>0</td><td>ENALE</td><td>FRW</td><td>0H</td><td>LB_BASE Enable: "1" to enable the aperture.</td></tr>
</table>

LB_MAP2: LOCAL BUS TO PCI BUS I/O ADDRESS MAP

Mnemonic: LB_MAP2
Offset: 66H
Size: 16 bits

LB_MAP2				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15-8	MAP_ADR	FRW	0H	Map Address: These bits correspond to bits AD[31:24] in PCI address space when a Local to PCI access is made. Address bits AD[23:2] are derived from the local bus itself. Access through this aperture results in an I/O cycle on the PCI bus.
7-0	-	R	0H	reserved

LB_SIZE: LOCAL BUS SIZE RREGISTER

Mnemonic:	LB_SIZE
Offset:	68H
Size:	32 bits

<table>
<tr><th colspan="5">LB_SIZE</th></tr>
<tr><th><i>Bits</i></th><th><i>Mnemonic</i></th><th><i>Type</i></th><th><i>Reset Value</i></th><th><i>Description</i></th></tr>
<tr><td>31-30</td><td>REGION_F</td><td>FRW</td><td>0H</td><td>Local Bus Width for addressregion 0xF0000000 to 0xFFFFFFFF:
<table>
<tr><th>REGION_EF</th><th>V350EPC (32 Bit mode) V360EPC</th><th>V350EPC in 16 Bit Slave Mode Only</th></tr>
<tr><td>00</td><td>32-bit</td><td>16-bit Unpacked</td></tr>
<tr><td>01</td><td>16-bit Packed</td><td>8-bit</td></tr>
<tr><td>10</td><td>8-bit</td><td>16-bit Packed</td></tr>
<tr><td>11</td><td>16-bit Unpacked</td><td>32-bit</td></tr>
</table></td></tr>
<tr><td>29-28</td><td>REGION_E</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0xE0000000 to 0xEFFFFFFF</td></tr>
<tr><td>27-26</td><td>REGION_D</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0xD0000000 to 0xDFFFFFFF</td></tr>
<tr><td>25-24</td><td>REGION_C</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0xC0000000 to 0xCFFFFFFF</td></tr>
<tr><td>23-22</td><td>REGION_B</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0xB0000000 to 0xBFFFFFFF</td></tr>
<tr><td>21-20</td><td>REGION_A</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0xA0000000 to 0xAFFFFFFF</td></tr>
<tr><td>19-18</td><td>REGION_9</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0x90000000 to 0x9FFFFFFF</td></tr>
<tr><td>17-16</td><td>REGION_8</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0x80000000 to 0x8FFFFFFF</td></tr>
<tr><td>15-14</td><td>REGION_7</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0x70000000 to 0x7FFFFFFF</td></tr>
<tr><td>13-12</td><td>REGION_6</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0x60000000 to 0x6FFFFFFF</td></tr>
<tr><td>11-10</td><td>REGION_5</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0x50000000 to 0x5FFFFFFF</td></tr>
<tr><td>9-8</td><td>REGION_4</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0x40000000 to 0x4FFFFFFF</td></tr>
<tr><td>7-6</td><td>REGION_3</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0x30000000 to 0x3FFFFFFF</td></tr>
<tr><td>5-4</td><td>REGION_2</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0x20000000 to 0x2FFFFFFF</td></tr>
<tr><td>3-2</td><td>REGION_1</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0x10000000 to 0x1FFFFFFF</td></tr>
<tr><td>1-0</td><td>REGION_0</td><td>FRW</td><td>0H</td><td>Local Bus Width for address region 0x00000000 to 0x0FFFFFFF</td></tr>
</table>

LB_IO_BASE: LOCAL BUS I/O BASE

Mnemonic:	LB_IO_BASE
Offset:	6EH
Size:	16 bits

<table>
<tr><th colspan="5">LB_IO_BASE</th></tr>
<tr><th><i>Bits</i></th><th><i>Mnemonic</i></th><th><i>Type</i></th><th><i>Reset Value</i></th><th><i>Description</i></th></tr>
<tr><td>15-0</td><td>ADR_BASE</td><td>FRW</td><td>0</td><td>Local to Internal Register Map Base Address: sets the base address for local bus side accesses to the EPC internal registers.
If the value of ADR_BASE matches that of local address bits 31:16 during the address phase of a local access then a match is detected. Since bits 31-16 are significant to the decoder, the size of the aperture is 64KB (only a small fraction of this is used).</td></tr>
</table>

FIFO_CFG: FIFO CONFIGURATION REGISTER

Mnemonic:	FIFO_CFG
Offset:	70H
Size:	16 bits

FIFO_CFG				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15-14	PBRST_MAX	FRW	0H	PCI Bus Maximum Burst Size : 00 = 4 Words 01 = 8 Words 10 = 16 Words 11 = 256 Words
13-12	PCI_WR_LB	FRW	0H	Write FIFO drain strategy for PCI Bus Write to Local Bus Aperture 0 and 1: 00 = Assert Local bus request immediately whenever the corresponding FIFO is not empty 01 = FIFO not empty and the PCI cycle filling it is finished[a] 10 = Assert Local bus request whenever the PCI bus to Local corresponding FIFO has 3 or more words of data pending 11 = Assert Local bus request whenever the PCI bus to Local FIFO has 3 or more words of data pending *or* the FIFO is not empty and the PCI cycle filling it is finished
11-10	PCI_RD_LB1[b]	FRW	0H	Read FIFO fill strategy for PCI Bus Read from Local Bus Aperture 1: 00 = Assert Local bus request whenever the corresponding read FIFO is not full (room for 1 or more words available). 01 = Assert Local bus request whenever the corresponding read FIFO is at most half full (room for 2 or more words available). 10 = Assert Local bus request whenever the corresponding FIFO is empty 11 = reserved
9-8	PCI_RD_LB0[b]	FRW	0H	FIFO control for PCI Bus Read from Local Bus Aperture 0: see description under PCI_RD_LB1, above.
7-6	LBRST_MAX	FRW	0H	Local Bus Maximum Burst Size: 00 =4 Words 01 =8 Words 10 =16 Words 11 =256 Words
5-4	LB_WR_PCI	FRW	0H	FIFO control for Local Bus Write to PCI Bus Aperture 0 and 1: 00 = Assert PCI bus request immediately whenever the corresponding FIFO is not empty 01 = FIFO not empty and the LB cycle filling it is finished 10 = Assert PCI bus request whenever the Local bus to PCI corresponding FIFO has 3 or more words of data pending 11 = Assert PCI bus request whenever the Local bus to PCI corresponding FIFO has 3 or more words of data pending *or* the FIFO is not empty and LB cycle filling it is finished
3-2	LB_RD_PCI1[b]	FRW	0H	FIFO control for Local Bus Read from PCI Bus Aperture 1: 00 = Assert PCI bus request whenever the corresponding read FIFO is not full (room for 1 or more words available) 01 = Assert PCI bus request whenever the corresponding read FIFO is at most half full (room for 2 or more words available). 10 = Assert PCI bus request whenever the corresponding FIFO is empty 11 = reserved
1-0	LB_RD_PCI0[b]	FRW	0H	FIFO control for Local Bus Read from PCI Bus Aperture 0: see description under LB_RD_PCI1, above. **(see foot notes on next page)**

[a] The cycle filling the FIFO could be a DMA read or slave write
[b] Has no effect on DMA transfers.

FIFO_PRIORITY: FIFO PRIORITY CONTROL REGISTER

Mnemonic:	FIFO_PRIORITY
Offset:	72H
Size:	16 bit

FIFO_PRIORITY				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15-13	-	R	0H	reserved
12	LOCAL	FRW	0H	Local Bus Request Priority: this controls the relative priority of pending transfers involving PCI to local bus. 0 = PCI write to local bus has priority over PCI read from local bus 1 = PCI read from local bus has priority over PCI write to local bus
11-10	LB_RD1	FRW	0H	Local Bus Read Flush Strategy for Aperture 1: this controls the condition that will cause Aperture 1 Local bus read prefetch FIFO to be flushed for the purpose of maintaining data coherency: 00 = Local bus to PCI writes never cause a flush[a] 01 = Flush at the end of a burst (don't keep extra data) 10 = Local bus to PCI write via aperture 1 will cause a flush (but not aperture 0) 11 = Any local bus to PCI write will cause a flush
9-8	LB_RD0	FRW	0H	Local Bus Read Flush Strategy for Aperture 0: this controls the condition that will cause aperture 0 Local bus read prefetch FIFO to be flushed for the purpose of maintaining data coherency: 00 = Local bus to PCI writes never cause a flush[a] 01 = Flush at the end of a burst (don't keep extra data) 10 = Local bus to PCI write via aperture 0 will cause a flush (but not aperture 1) 11 = Any local bus to PCI write will cause a flush
7-5	-	R	0H	reserved
4	PCI	FRW	0H	PCI Bus Request Priority: this controls the relative priority of pending transfers involving local bus to PCI bus. 0 = Local bus write to PCI has priority over Local bus read from PCI 1 = Local bus read from PCI has priority over Local bus write to PCI
3-2	PCI_RD1	FRW	0H	PCI Read Flush Strategy for Aperture 1: this controls the condition that will cause Aperture 1 PCI read prefetch FIFO to be flushed for the purpose of maintaining data coherency: 00 = PCI to local bus writes never cause a flush[a] 01 = Flush at the end of a burst (don't keep extra data) 10 = PCI to local bus write via aperture 1 will cause a flush (but not writes to aperture 0) 11 = Any PCI to local bus write will cause a flush

1-0	PCI_RD0	FRW	0H	PCI Read Flush Strategy for Aperture 0: this controls the condition that will cause aperture 0 PCI read prefetch FIFO to be flushed for the purpose of maintaining data coherency: 00 = PCI to local bus writes never cause a flush[a] 01 = Flush at the end of a burst (don’t keep extra data) 10 = PCI to local bus write via aperture 0 will cause a flush (but not writes to aperture 1) 11 = Any PCI to local bus write will cause a flush

[a] Only this option should be chosen when prefetching is disabled for the aperture. When prefetching is disabled, there is never any prefetch data to flush anyway.

FIFO_STAT: FIFO STATUS REGISTER

Mnemonic:	FIFO_STAT
Offset:	74H
Size:	16 bits

FIFO_STAT				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15-14	-	R	0H	reserved
13-12	L2P_WR	R	0H	Status of the Local-to-PCI write FIFO:[a] 00 = Empty 01 = (While Filling) One or more words has been placed in the FIFO (While Draining) One data word remaining in the FIFO to be written to the PCI bus 10 = FIFO is completely full 11 = FIFO has room for only one more word
11-10	L2P_RD1	R	0H	PCI Bus Read from Local Bus Aperture 1 FIFO Status: 00 = Between empty and full 01 = Empty 10 = Full 11 = Between empty and full
9-8	L2P_RD0	R	0H	PCI Bus Read from Local Bus Aperture 0 FIFO Status: 00 = Between empty and full 01 = Empty 10 = Full 11 = Between empty and full
7-6	-	R	0H	reserved
5-4	P2L_WR	R	0H	Status of the PCI-to-Local write FIFO:[b] 00 = Empty 01 = (While Filling) One or more words has been placed in the FIFO (While Draining) One data word remaining in the FIFO to be written to the PCI bus 10 = FIFO is completely full 11 = FIFO has room for only one more word
3-2	P2L_RD1	R	0H	Local Bus Read from PCI Bus Aperture 1 FIFO Status: 00 = Between empty and full 01 = Empty 10 = Full 11 = Between empty and full
1-0	P2L_RD0	R	0H	Local Bus Read from PCI Bus Aperture 0 FIFO Status: 00 = Between empty and full 01 = Empty 10 = Full 11 = Between empty and full

[a] The transitions for the bits in this field follows a hysterisis curve depending on whether the FIFO has reached certain levels of "fullness". Please see the "Bridge Operation" section for more detail.
[b] The transitions for the bits in this field follows a hysterisis curve depending on whether the FIFO has reached certain levels of "fullness". Please see the "Bridge Operation" section for more detail.

LB_ISTAT: LOCAL BUS INTERRUPT CONTROL AND STATUS REGISTER

Mnemonic:	LB_ISTAT
Offset:	76H
Size:	8 bits

LB_ISTAT				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
7	MAILBOX[a]	R	0H	1 = an interrupt has been requested by one or more of the mailbox registers 0 = no mailbox interrupts pending
6	PCI_RD	FRW	0H	1 = Target Abort or lack of Device Select has been seen during a local bus to PCI bus read access and an interrupt request for such events has been enabled; clear by writing "0" 0 = no local bus read of PCI space error interrupt request is pending.
5	PCI_WR	FRW	0H	1 = Target Abort or lack of Device Select is seen during a local bus to PCI bus write access and an interrupt request for such events has been enabled; clear by writing "0" 0 = no local bus write to PCI space error interrupt request is pending.
4	PCI_INT[b]	FRW	0H	1 = a PCI interrupt pin has requested an interrupt 0 = no pending PCI interrupt events
3	PCI_PERR	RW	0H	PCI Parity Error Interrupt: This bit is set in response to parity error seen on the PCI bus as a result of the VxxxEPC acting as either a master or a slave for the cycle.
2	I2O_QWR[a]	RW	0H	I2O Inbound Post Queue Write Interrupt: This bit is set when 3 conditions are met: I2O is enabled, the corresponding bit in LB_IMASK is enabled and the inbound post list is written. Cleared by writing ‘0’.
1	DMA1	FRW	0H	1 = DMA channel 1 has requested an interrupt;clear by writing"0" 0 = DMA channel 1 has not requested an interrupt.
0	DMA0	FRW	0H	1 = DMA channel 0 has requested an interrupt;clear by writing "0" 0 = DMA channel 0 has not requested an interrupt.

[a] This bit is a logical OR of all of the mailbox interrupt requests. It is only cleared when all of the individual mailbox interrupt requests have been cleared via reading or writing the applicable mailbox register. See the "Mailbox Register" section of the "Bridge Operation" chapter for more information. This bit is FRW on Revision B0/B1 and must be cleared by writing ’0’ after first clearing all mailbox interrupt sources.

[b] Note: All writable status bits are cleared by writing ’0’. Writing ’1’ has no effect.

LB_IMASK: LOCAL BUS INTERRUPT MASK REGISTER

Mnemonic:	LB_IMASK
Offset:	77H
Size:	8 bits

LB_IMASK				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
7	MAILBOX	FRW	0H	Global Mailbox Interrupt Enable: 1 = mailbox interrupts are enabled 0 = all mailbox interrupts are masked
6	PCI_RD	FRW	0H	PCI Read Error Interrupt Enable: 1 = enable local interrrupt requests for PCI read errors 0 = mask local interrrupt requests for PCI read errors
5	PCI_WR	FRW	0H	PCI Write Error Interrupt Enable: 1 = enable local interrrupt requests for PCI write errors 0 = mask local interrrupt requests for PCI write errors
4	PCI_INT	FRW	0H	Global PCI Interrupt to Local Interrupt Enable (not in Rev A): 1 = enable PCI interrupt requests to request a local interrupt 0 = mask PCI interrupt requests to request a local interrupt
3	PCI_PERR	RW	0H	PCI Parity Error Interrupt Enable: When enabled (1) the corresponding bit in LB_ISTAT is set in response to a parity error event seen on the PCI bus. In order for a PCI parity event to be detected one or more of the MASTER_PI and/or the SLAVE_PI bits in PCI_INT_CFG must be enabled in addition to this bit.
2	I2O_QWR	RW	0H	I2O Inbound Post Queue Write Interrupt Enable: Set (1) to enable inbound post queue write cycles to generate interrupts on the Local Bus.
1	DMA1	FRW	0H	DMA Channel 1 Interrupt Enable: 1 = enable DMA Channel 1 interrupt requests 0 = mask DMA Channel 1 interrupt requests
0	DMA0	FRW	0H	DMA Channel 0 Interrupt Enable: 1 = enable DMA Channel 0 interrupt requests 0 = mask DMA Channel 0 interrupt requests

SYSTEM REGISTER

Mnemonic:	SYSTEM
Offset:	78H
Size:	16 bits

SYSTEM				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15	RST_OUT	FR	0H	Reset Output Control: When set to ‘1’ the reset output (see RDIR pin description) is de-asserted.
14	LOCK	FR	0H	Lock Register Contents: When set to ‘1’ the SYSTEM register becomes unwritable. LOCK can only be cleared by writing the SYSTEM register with 0xA05F[a].
13	SPROM_EN	FR	0H	Serial PROM Software Access Enable. 1 = SCL/Local parity error pin functions as SCL 0 = SCL/Local parity error pin functions as local parity error
12	SCL	FR	0H	Serial PROM Clock Output. When SPROM_EN is enabled (‘1’) then this bit controls the state of the SCL pin.
11	SDA_OUT	FR	0H	Serial PROM Data Output. When SPROM_EN is enabled (‘1’) then this bit controls the state of the SDA pin. When SDA_OUT is ‘0’ then the SDA pin will be driven low. For SDA_OUT = ‘1’ the SDA pin floats to high impedance.
10	SDA_IN	R	X	Serial PROM Data In. Reads back the SDA input directly from the pin.
9	POE	FR	0H	Local Bus Parity: 1=odd parity 0=even parity
8	FAST_REQ	FR	0H	0 = bus follows strict Am29030 protocol 1 = bus follows high-performance Am29K protocol This bit has no affect on devices other than the V292EPC.
7	-	R	0H	reserved
6	LB_RD_PCI1	W	0H	1 = Local Read from PCI Bus (Aperture 1) FIFO Flush [b] 0 = no operation
5	LB_RD_PCI0	W	0H	1 = Local Read from PCI Bus (Aperture 0) FIFO Flush [b] 0 = no operation
4	LB_WR_PCI	W	0H	1 = Local to PCI Write FIFO Flush [b] 0 = no operation
3	-	R	0H	reserved
2	PCI_RD_LB1	W	0H	1 = PCI Read from Local Bus (Aperture 1) FIFO Flush [b] 0 = no operation
1	PCI_RD_LB0	W	0H	1 = PCI Read from Local Bus (Aperture 0) FIFO Flush [b] 0 = no operation
0	PCI_WR_LB	W	0H	1 = PCI to Local Write FIFO Flush [b] 0 = no operation

[a] Writing 0xA05F to un-lock the system register will not overwrite the current System register values.

[b] LOCK bit in SYSTEM register must be set to "0" to write this bit.

LB_CFG REGISTER: LOCAL BUS CONFIGURATION REGISTER[a]

Mnemonic:	LB_CFG
Offset:	7AH
Size:	16 bits

LB_CFG				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15-14	-	R	0H	reserved
13	TO_LENGTH	FRW		Local Bus Time-out Time Constant: '1' = 256 cycles, '0' = 64 cycles
12	-	R	0H	reserved
11	PPC_RDY	FRW	0H	Power PC Ready: this bit is defined only for the V292EPC and determines how the RDY signal operates when a local bus master (such as a PPC403Gx) accesses the V292EPC for read cycles. When disabled (0) the RDY signal operates normally. However, when enabled (1) the relationship of read data to the RDY signal is modified so that data will exist for one cycle after RDY (normally valid data would be seen at the same time as RDY is seen). This allows the V292EPC to be used with the PPC403Gx with only a small single programmable logic device. No address/data path registers are required.
10	LB_INT	FRW	0H	Local Bus Interrupt Enable: When this bit is enabled and the PCI_RD and/or PCI_WR bits are enabled in LB_IMASK, a time-out event will cause the local bus interrupt to be asserted via the LB_ISTAT register bits PCI_RD (for time-out on a read) or PCI_WR (for time-out on a write)
9	ERR_EN	FRW	0H	BTERM/ERR Enable: When enabled ('1') the V962EPC will assert the local bus BTERM(V961EPC, V962EPC) or ERR(V292EPC) signal for one clock whenever a time-out occurs. A time-out event occurs when a request to the EPC is outstanding for longer than the value determined by the TO_LENGTH register bit.
8	RDY_EN	FRW	0H	Ready Enable: When enabled ('1') the EPC will assert the local bus READY or RDY signal for one clock whenever a time-out occurs.
7[b]	BE_IMODE	FRW	0H	Byte Enable Input Mode: this bit is defined only for the V292EPC and determines how the BWE[3:0] signals operate as inputs. When this bit is clear (0) the local bus slave controller on the EPC will assume that any read access to the V292EPC by an external master device is 32 bits (all bytes are enabled). When set (1) the V292EPC slave controller will treat the BWE[3:0] as byte enables for both read and write.
6[b]	BE_OMODE	FRW	0H	Byte Enable Output Mode: this bit is defined only for the V292EPC and determines how the BWE[3:0] signals operate as outputs. When this bit is clear (0) the local bus master controller on the EPC will drive the BWE[3:0] active only during write cycles and remain de-asserted for read cycles. When set (1) the V292EPC will assert the appropriate byte enable information onto the BWE[3:0] during both read and write cycles.
5[b]	ENDIAN	FRW	0H	Endian Mode: This determines where 8 an 16-bit data is connected to the 32-bit local data bus. It also determines the byte order for 8/16 bit operations.
4-0	-	R	0H	reserved

PCI_CFG: PCI BUS CONFIGURATION REGISTER[a]

Mnemonic:	PCI_CFG
Offset:	7CH
Size:	16 bits

PCI_CFG				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15	I2O_EN	FRW	0H	I2O Enable
14	IO_REG_DIS	FRW	0H	Disable PCI_IO_BASE register: when set (1) the contents of PCI_IO_BASE register will read back ‘0’ although the register may actually contain non-zero data. The register remains writeable and the decode function of the chip will remain intact.
13	IO_DIS	FRW	0H	Disable PCI_IO_BASE Decoder: when set (1) PCI_IO_BASE will not respond to PCI cycles.
12	EN3V	FR	0H	Enable I/O buffers for 3.3V operation.
11-10	-	R	0H	reserved
9-8	AD_LOW	FRW	0H	PCI AD[1:0]. Override Value: When one of the LB_MAP registers is programmed with AD_LOW_EN set (1) then the value in this register is used to generate AD[1:0] instead of the normal value (“00” for all except I/O cycles where the byte enables determine the value).
7-5	DMA_RTYPE	FRW	0H	DMA Read from PCI Bus Command Type: determines the PCI command type applied to the C/BE#(3:1) outputs for a PCI read cycle from the DMA controller. C/BE# bit 0 will be driven ‘0’ for all write cycles and thus there is no corresponding register bit. Writing DMA_RTYPE “000” will cause the value 011 to be written instead. This results in a “memory read” command type.
4	-	R	0H	reserved
3-1	DMA_WTYPE	FRW	0H	DMA Write to PCI Bus Command Type: determines the PCI command type applied to the C/BE[3:1] outputs for a PCI write cycle from the DMA controller. C/BE bit 0 will be driven ‘1’ for all write cycles and thus there is no corresponding register bit. Writing DMA_WTYPE “000” will cause the value 011 to be written instead. This results in a “memory write” command type.
0	-	R	0H	reserved

DMA_PCI_ADR: PCI DMA ADDRESS REGISTERS

Mnemonic:	DMA_PCI_ADDR0, 1
Offset:	80H, 90H
Size:	32 bits

DMA_PCI_ADDR0, 1				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
31-2	ADR	RW	0H	PCI Byte Address
1-0		R	0H	These low address bits read back zero since all DMA transfers are 32 bit aligned

DMA_LOCAL_ADR: LOCAL DMA ADDRESS REGISTERS

Mnemonic:	DMA_LOCAL_ADDR0, 1
Offset:	84H, 94H
Size:	32 bits

DMA_LOCAL_ADDR0, 1				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
31-2	ADR	RW	0H	Local Byte Address
1-0		R	0H	These low address bits read back zero since all DMA transfers are 32 bit aligned

DMA_LENGTH0, 1: DMA TRANSFER LENGTH REGISTER 0, 1

Mnemonic:	DMA_LENGTH0, 1
Offset:	88H, 98H
Size:	24 bits

DMA_LENGTH0, 1				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
23	DREQ_EN	RW	0H	External DMA Request Enable: When set (1) the DMA will be throttled by to the state of the INTC# input pin (DMA Channel 0) or INTD# input pin (DMA Channel 1). The corresponding pin must be low (0) to allow the DMA to fetch the data source.
22	INTR_EN	RW	0H	Interrupt on Link Complete: When set (1) an internal interrupt from the DMA controller will generated whenever the data transfer portion of a link is complete. The internal DMA interrupt can be routed to PCI or local interrupt outputs by enabling them in the PCI_INT_CFG and/or LB_IMASK regiters. An internal interrupt is always generated upon chain completion when the DMA_IPR bit is cleared by the hardware.
21-20	-	R	0H	reserved
19-0	COUNT	RW	0H	Transfer Count Remaining. This register holds the initial transfer count (in 32-bit words) and is updated with the remaing count after each DMA transfer.

DMA_CSR: DMA CONTROL AND STATUS REGISTERS

Mnemonic: DMA_CSR0, 1
Offset: 8BH, 9BH
Size: 8 bits

<table>
<tr><th colspan="5">DMA_CSR0, 1</th></tr>
<tr><th>Bits</th><th>Mnemonic</th><th>Type</th><th>Reset Value</th><th>Description</th></tr>
<tr><td>7</td><td>CHAIN</td><td>RW</td><td>0H</td><td>1 = enable DMA chaining for this transfer
0 = disable DMA chaining for this transfer</td></tr>
<tr><td>6</td><td>CLR_LEN</td><td>RW</td><td>0H</td><td>Clear Length: when set (1), the DMA_LENGTH value in the memory based descriptor will be cleared after the transfer is complete.</td></tr>
<tr><td>5</td><td>PRIORITY</td><td>RW</td><td>0H</td><td>Controls the relative priority of DMA channels. See "DMA Controller" chapter. (Not in Rev A)</td></tr>
<tr><td>4</td><td>DIRECTION</td><td>RW</td><td>0H</td><td>DMA Direction:
0 = Local to PCI
1 = PCI to Local</td></tr>
<tr><td>3-2</td><td>SWAP</td><td>RW</td><td>0H</td><td>Byte Swap Control: Selects byte order conversion options:[a]
<table>
<tr><th></th><th>SWAP</th><th>D[31:24]</th><th>D[23:26]</th><th>D[15:8]</th><th>D[7:0]</th></tr>
<tr><td>no swap, 32 bit</td><td>00</td><td>Q[31:24]</td><td>Q[23:16]</td><td>Q[15:8]</td><td>Q[7:0]</td></tr>
<tr><td>16 bit</td><td>01</td><td>Q[15:8]</td><td>Q[7:0]</td><td>Q[31:24]</td><td>Q[23:16]</td></tr>
<tr><td>8 bit</td><td>10</td><td>Q[7:0]</td><td>Q[15:8]</td><td>Q[23:16]</td><td>Q[31:24]</td></tr>
<tr><td>reserved</td><td>11</td><td></td><td></td><td></td><td></td></tr>
</table></td></tr>
<tr><td>1</td><td>ABORT</td><td>W</td><td>0H</td><td>1 = immediately abort current DMA transfer in process[a]
0 = no operation</td></tr>
<tr><td>0</td><td>DMA_IPR</td><td>RW</td><td>0H</td><td>DMA Initiate Process:
Write 1 = begin DMA operation
Write 0 = no operation
Read 1 = DMA in Progress
Read 0 = DMA Idle
Automatically cleared when the transfer count expires and there are no further chains to process.</td></tr>
</table>

a. Writing to DMA_CSRx with the ABORT bit set will cause all other bits in the register to maintain their previous value (they are not written). The transfer can be restarted by setting DMA_IPR again using a read-modify-write to maintain the contents of the other register bits.

DMA_CTLB_ADR: DMA CONTROL BLOCK ADDRESS REGISTER 0,1

Mnemonic: DMA_CTLB_ADDR0, 1
Offset: 8CH, 9CH
Size: 32 bits

DMA_CTLB_ADDR0, 1

Bits	*Mnemonic*	*Type*	*Reset Value*	*Description*
31-4	CTLB_ADR	RW	0H	DMA control block address. Address of the first control block in a DMA chain. Must be aligned to a 16 byte boundary and reside in Local memory.
3-0	-	R	0H	reserved

DMA_DELAY: DMA DESCIPTOR DELAY

Mnemonic: DMA_DELAY
Offset: E0H
Size: 8 bits

DMA_DELAY

Bits	*Mnemonic*	*Type*	*Reset Value*	*Description*
7-0	DELAY	RW	0H	Determines the delay between the completion of processing a DMA descriptor and the loading of the next DMA descriptor.

MAIL_DATA0-15: MAILBOX DATA REGISTER 0-15

Mnemonic: MAIL_DATA
Offset: C0H - CFH (see register map)
Size: 8 bits

MAIL_DATA0-15

Bits	*Mnemonic*	*Type*	*Reset Value*	*Description*
7-0	DATA	RW	0H	Mailbox Data (mapped into local bus address space): Application specific, software defined data register.

PCI BUS MAILBOX WRITE INTERRUPT CONTROL REGISTER

Mnemonic:	PCI_MAIL_IEWR
Offset:	D0H
Size:	16 bits

PCI_MAIL_IEWR				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15	EN15	RW	0H	1 = enable local interrupts on PCI bus writes to mailbox 15 0 = disable local interrupts on PCI bus writes to mailbox 15
14	EN14	RW	0H	Same as above for mailbox 14.
13	EN13	RW	0H	Same as above for mailbox 13.
12	EN12	RW	0H	Same as above for mailbox 12.
11	EN11	RW	0H	Same as above for mailbox 11.
10	EN10	RW	0H	Same as above for mailbox 10.
9	EN9	RW	0H	Same as above for mailbox 9.
8	EN8	RW	0H	Same as above for mailbox 8.
7	EN7	RW	0H	Same as above for mailbox 7.
6	EN6	RW	0H	Same as above for mailbox 6.
5	EN5	RW	0H	Same as above for mailbox 5.
4	EN4	RW	0H	Same as above for mailbox 4.
3	EN3	RW	0H	Same as above for mailbox 3.
2	EN2	RW	0H	Same as above for mailbox 2.
1	EN1	RW	0H	Same as above for mailbox 1.
0	EN0	RW	0H	Same as above for mailbox 0.

PCI BUS MAILBOX READ INTERRUPT CONTROL REGISTER

Mnemonic:	PCI_MAIL_IERD
Offset:	D2H
Size:	16 bits

PCI_MAIL_IERD				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15	EN15	RW	0H	1 = enable local interrupts on PCI bus read from mailbox 15 0 = disable local interrupts on PCI bus read from mailbox 15
14	EN14	RW	0H	Same as above for mailbox 14.
13	EN13	RW	0H	Same as above for mailbox 13.
12	EN12	RW	0H	Same as above for mailbox 12.
11	EN11	RW	0H	Same as above for mailbox 11.
10	EN10	RW	0H	Same as above for mailbox 10.
9	EN9	RW	0H	Same as above for mailbox 9.
8	EN8	RW	0H	Same as above for mailbox 8.
7	EN7	RW	0H	Same as above for mailbox 7.
6	EN6	RW	0H	Same as above for mailbox 6.
5	EN5	RW	0H	Same as above for mailbox 5.

PCI_MAIL_IERD (cont'd)				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
4	EN4	RW	0H	Same as above for mailbox 4.
3	EN3	RW	0H	Same as above for mailbox 3.
2	EN2	RW	0H	Same as above for mailbox 2.
1	EN1	RW	0H	Same as above for mailbox 1.
0	EN0	RW	0H	Same as above for mailbox 0.

LOCAL BUS MAILBOX WRITE INTERRUPT CONTROL REGISTER

Mnemonic:	LB_MAIL_IEWR
Offset:	D4H
Size:	16 bits

LB_MAIL_IEWR				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15	EN15	RW	0H	1 = enable PCI interrupts on local bus writes to mailbox 15 0 =disable PCI interrupts on local bus writes to mailbox 15
14	EN14	RW	0H	Same as above for mailbox 14.
13	EN13	RW	0H	Same as above for mailbox 13.
12	EN12	RW	0H	Same as above for mailbox 12.
11	EN11	RW	0H	Same as above for mailbox 11.
10	EN10	RW	0H	Same as above for mailbox 10.
9	EN9	RW	0H	Same as above for mailbox 9.
8	EN8	RW	0H	Same as above for mailbox 8.
7	EN7	RW	0H	Same as above for mailbox 7.
6	EN6	RW	0H	Same as above for mailbox 6.
5	EN5	RW	0H	Same as above for mailbox 5.
4	EN4	RW	0H	Same as above for mailbox 4.
3	EN3	RW	0H	Same as above for mailbox 3.
2	EN2	RW	0H	Same as above for mailbox 2.
1	EN1	RW	0H	Same as above for mailbox 1.
0	EN0	RW	0H	Same as above for mailbox 0.

LOCAL BUS MAILBOX READ INTERRUPT CONTROL REGISTER

Mnemonic:	LB_MAIL_IERD
Offset:	D6H
Size:	16 bits

LB_MAIL_IERD				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15	EN15	RW	0H	1 = enable PCI interrupts on local bus reads from mailbox 15 0 = disable PCI interrupts on local bus reads from mailbox 15
14	EN14	RW	0H	Same as above for mailbox 14.
13	EN13	RW	0H	Same as above for mailbox 13.
12	EN12	RW	0H	Same as above for mailbox 12.
11	EN11	RW	0H	Same as above for mailbox 11.
10	EN10	RW	0H	Same as above for mailbox 10.
9	EN9	RW	0H	Same as above for mailbox 9.
8	EN8	RW	0H	Same as above for mailbox 8.
7	EN7	RW	0H	Same as above for mailbox 7.
6	EN6	RW	0H	Same as above for mailbox 6.
5	EN5	RW	0H	Same as above for mailbox 5.
4	EN4	RW	0H	Same as above for mailbox 4.
3	EN3	RW	0H	Same as above for mailbox 3.
2	EN2	RW	0H	Same as above for mailbox 2.
1	EN1	RW	0H	Same as above for mailbox 1.
0	EN0	RW	0H	Same as above for mailbox 0.

MAIL_WR_STAT: MAILBOX WRITE INTERRUPT STATUS

Mnemonic:	MAIL_WR_STAT
Offset:	D8H
Size:	16 bits

MAIL_WR_STAT				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15	WR_STAT15	RW	0H	1 = Mailbox 15 has requested a PCI or local write interrupt 0 = Mailbox 15 has not requested a PCI or local write interrupt Cleared by writing ‘1’. Writing ‘0’ has no effect
14	WR_STAT14	RW	0H	same as above for mailbox 14
13	WR_STAT13	RW	0H	same as above for mailbox 13
12	WR_STAT12	RW	0H	same as above for mailbox 12
11	WR_STAT11	RW	0H	same as above for mailbox 11
10	WR_STAT10	RW	0H	same as above for mailbox 10
9	WR_STAT9	RW	0H	same as above for mailbox 9
8	WR_STAT8	RW	0H	same as above for mailbox 8
7	WR_STAT7	RW	0H	same as above for mailbox 7
6	WR_STAT6	RW	0H	same as above for mailbox 6

MAIL_WR_STAT (cont'd)				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
5	WR_STAT5	RW	0H	same as above for mailbox 5
4	WR_STAT4	RW	0H	same as above for mailbox 4
3	WR_STAT3	RW	0H	same as above for mailbox 3
2	WR_STAT2	RW	0H	same as above for mailbox 2
1	WR_STAT1	RW	0H	same as above for mailbox 1
0	WR_STAT0	RW	0H	same as above for mailbox 0

MAIL_RD_STAT: MAILBOX READ INTERRUPT STATUS

Mnemonic:	MAIL_RD_STAT
Offset:	DAH
Size:	16 bits

MAIL_RD_STAT				
Bits	***Mnemonic***	***Type***	***Reset Value***	***Description***
15	RD_STAT15	RW	0H	1 = Mailbox 15 has requested a PCI or local read interrupt 0 = Mailbox 15 has not requested a PCI or local read interrupt Cleared by writing '1'. Writing '0' has no effect
14	RD_STAT14	RW	0H	same as above for mailbox 14
13	RD_STAT13	RW	0H	same as above for mailbox 13
12	RD_STAT12	RW	0H	same as above for mailbox 12
11	RD_STAT11	RW	0H	same as above for mailbox 11
10	RD_STAT10	RW	0H	same as above for mailbox 10
9	RD_STAT9	RW	0H	same as above for mailbox 9
8	RD_STAT8	RW	0H	same as above for mailbox 8
7	RD_STAT7	RW	0H	same as above for mailbox 7
6	RD_STAT6	RW	0H	same as above for mailbox 6
5	RD_STAT5	RW	0H	same as above for mailbox 5
4	RD_STAT4	RW	0H	same as above for mailbox 4
3	RD_STAT3	RW	0H	same as above for mailbox 3
2	RD_STAT2	RW	0H	same as above for mailbox 2
1	RD_STAT1	RW	0H	same as above for mailbox 1
0	RD_STAT0	RW	0H	same as above for mailbox 0

QBA_MAP : LOCATING THE QUEUE IN LOCAL MEMORY[a]

Mnemonic: QBA_MAP
Offset: DCH
Size: 32 bits

<table>
<tr><th colspan="5">QBA_MAP</th></tr>
<tr><th><em>Bits</em></th><th><em>Mnemonic</em></th><th><em>Type</em></th><th><em>Reset Value</em></th><th><em>Description</em></th></tr>
<tr><td>31-20</td><td>BASE</td><td>RW</td><td>0H</td><td>Queue Base Address: These bits are used to generate bits 31-20 of all Inbound/Outbound pointers.egister bits so that lower bits of the decode are masked off.</td></tr>
<tr><td>19-10</td><td>-</td><td>R</td><td>0H</td><td>reserved</td></tr>
<tr><td>10-8</td><td>QSIZE</td><td>RW</td><td>0H</td><td>Queue Size: The size of the aperture is determined as follows::
<table>
<tr><th>QSIZE</th><th>Size</th><th>Auto Increment Mask</th></tr>
<tr><td>000</td><td>4K entries (16K Bytes)</td><td>0x3FFC</td></tr>
<tr><td>001</td><td>8K entries (32K Bytes)</td><td>0x7FFC</td></tr>
<tr><td>010</td><td>16K entries (64K Bytes)</td><td>0xFFFC</td></tr>
<tr><td>011</td><td>32K entries (128K Bytes)</td><td>0x1FFFC</td></tr>
<tr><td>100</td><td>64K entries (256K Bytes)</td><td>0x3FFFC</td></tr>
<tr><td>others</td><td colspan="2">reserved</td></tr>
</table>
</td></tr>
<tr><td>7-1</td><td>-</td><td>R</td><td>0H</td><td>reserved</td></tr>
<tr><td>0</td><td>ONLINE</td><td>RW</td><td>0H</td><td>I2O Device On Line: when clear (0), the VxxxEPC will return 0xFFFFFFFF when the inbound and outbound ports are read. This bit should only be enabled after the queues have been initialized by the local processor and it is ready to accept an inbound MFA. This bit must be clear in order for the local processor to modify the contents of the OFL_HEAD, OPL_TAIL, IPL_HEAD and IFL_TAIL registers.</td></tr>
</table>

Glossary

This glossary contains terms used in EPC User’s Manual. For reference to more information about a term, please refer to index and PCI 2.1 specification.

Burst - Bus protocol that allows a bus master to request more than one data transfer for an access. Typically, the data is sequential, sequential over a modulo boundary or cache line toggle (such as the 486).

Deadlock - The condition in which one processor (or master device) attempts to access a resource that is tightly coupled to a second processor (or master device) that is also in a state of attempting to access a resource local to the first processor.

DMA - Direct Memory Access. A DMA controller allows the CPU to continue operation while the EPC controls block transfer between Local and PCI address space.

Dynamic Bandwidth Allocation™ - A technique that allows a single FIFO to be dynamically shared for multiple purposes and also negates the need to require a bus retry each time a non-sequential address is begun.

Endian - The organization of sub-word data within the physical data bus. Little Endian processors address the first byte of a 32-bit word on the least significant data lines. Big Endian processors address the first byte of a 32-bit word on the most significant data lines. The PCI bus is strictly little endian and the internal registers of the EPC reflect this fact.

Wait State - A processor clock cycle in which no data transfer occurs although data transfer has been requested. Used to hold off a requesting master until data from memory or I/O is ready.

Word - The native bus size of the system. For this document, it is 32 bits which is the size of the PCI bus.

Index

Index